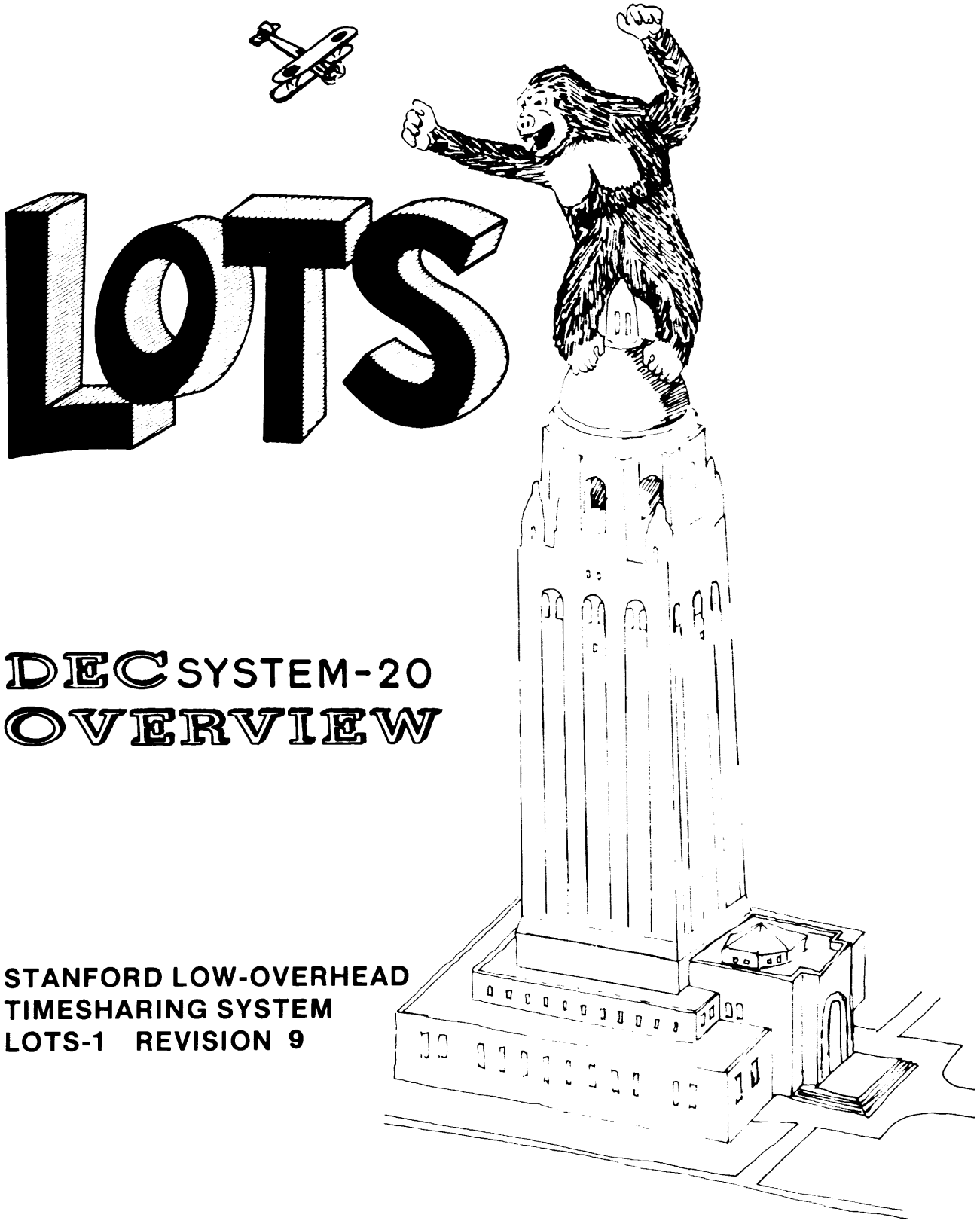


DEC SYSTEM-20 OVERVIEW

STANFORD LOW-OVERHEAD
TIMESHARING SYSTEM
LOTS-1 REVISION 9

LOTS DECSYSTEM-20 Overview Manual

LOTS Operating Note 1, Revision 9
Stanford Low Overhead Timesharing System
26 September 1984

This manual was prepared using SCRIBE, the document compiler created by Brian K. Reid, and typeset on the Xerox Dover printer. We thank the staff and management of the Stanford Computer Science Department Computer Facilities for their assistance in the preparation of this document, and for making available an environment in which computer assisted document preparation is not only possible but almost easy.

Preface

This manual tries to introduce the reader to some important computer concepts and their application to the DECSYSTEM-20 computers at Stanford. It includes information that is especially useful for people who are new to computing. Although there is a great deal of material here, it is not necessary to learn all of it. However, the more you know, the better you will understand the use of the computer.

The greatest difficulty people have in using the computer is their refusal to read and execute instructions exactly as they are given. The computer expects precise and specific commands from its users. If the commands the computer receives are wrong, the computer may fail to understand. Worse, it may carry out the mistaken directions. In dealing with the computer, not everything is done by rote; there is ample room for creativity. However, as the fortune cookie says, "He who has imagination but no learning has wings but no feet." Before attempting to "fly" on the computer, use this manual to learn where to put your feet.

Although this manual is called the *LOTS Overview Manual*, it has been written with an eye towards being relevant to all the DECSYSTEM-20 computers at Stanford. As of this writing there are nine DEC-20's at Stanford: LOTSA, LOTSB, and LOTSC at the Low Overhead Time-Sharing (LOTS) Computer Facility, How and Why at the Graduate School of Business (GSB) Computer Facility, Score at the Computer Science Department, Sierra at the Electrical Engineering Department, SUMEX-AIM at the Stanford Medical Center, and Turing at the Center for the Study of Language and Information (CSLI).

The manual has four main sections. The first section, *Getting Started on the DEC-20* is a general introduction to the DEC-20 computer and how to use it. The second section, *Using the EXEC Command Language* describes how to use the program called EXEC, which provides many general utilities. The third section, *An Overview of the Stanford DEC-20's* describes some of the policies and operating idiosyncracies of the various Stanford computer facilities. The fourth section, *Program Descriptions*, consists of a set of descriptions of a number of common programs on the DEC-20. It is organized by topic; for example, the section on magnetic tapes (section 31, page 229) contains a discussion of all of the commonly used tape handling programs.

It is not necessary to read all of one section before looking at other parts. However, most people will find it useful to know nearly all of the material found in the *Getting Started on the DEC-20* section.

One convention that is used throughout to make the manual more understandable is that in examples, anything that the user (you) types is underlined. Those things typed by the computer are not underlined.

The *LOTS Overview Manual* was created and is maintained by the staff and volunteers of the LOTS Facility. There are many shortcomings in this manual; it is difficult to include enough material without introducing irrelevancies. Specific suggestions for improving the manual will be most welcome. Please communicate any suggestions you have to the LOTS staff.

Good luck! And please, remember, follow the directions.

Table of Contents

Part I

Getting Started on the DEC-20

1. Getting Acquainted

People communicate with the computer by means of computer terminals. Terminals may be located in terminal clusters, such as the LOTS clusters in the CERAS and Terman buildings, or they may be located in people's offices or homes.

There are three ways a terminal may be hooked up to a computer. If a terminal is directly connected to a particular computer, i.e., you can use that terminal to talk only to that computer, it is referred to as a hardwired terminal. If the terminal communicates with the computer over the telephone lines, it is called a dial-up terminal. When a terminal is attached to the campus wide communications network (referred to as the Ethernet), it is called an Ethernet terminal. In general it is possible to use an Ethernet terminal to connect to any Stanford computer that is on the Ethernet.

We will first describe some of the common characteristics of a terminal and then give examples on how to use terminals when they are hooked up to computers in different ways.

1.1. Learning to Use the Terminal

There is an on-line tutorial designed to teach you some of the fundamentals of working on the computer. It is called NOVICE. To run the program, simply type NOVICE to the @ prompt and press the RETURN key:

```
@novice
```

It would be a good idea to both read this section and run NOVICE. There is some overlap, but this section covers material not discussed by NOVICE, and NOVICE demonstrates things to you directly instead of just having you read.

Since you "talk" to the computer by means of a computer terminal, you must become familiar with terminals and how to use them. Terminals have two main parts. One part is a TV screen that displays what you type and what the computer types in response. An underline or a bright square on the TV screen marks the "cursor", the position where the next character that is typed will go. The second part of the terminal is a keyboard that is similar to the keyboard on a typewriter. Study the terminal keyboard carefully to become familiar with the special keys there.

The SHIFT key is used to form uppercase letters, just as on an ordinary typewriter. Normally when you type a key such as "H", the character you get is lowercase, i.e. "h". Hold down SHIFT while typing a letter to get the uppercase letter, "H". Some keys have more than one symbol. For example, there is a key labeled with "!" above "1". To get a "1", type this key; to get "!", hold down SHIFT and type this key. In cases where there might be some confusion, we refer to the "!" character as SHIFT/1.

The DELETE key is one of the most useful things about using computer terminals. When you make a typing mistake, you may use the DELETE key to erase the last character on the current line. If you type DELETE again, another character goes away. When you discover an error in the current line you are typing, use DELETE to erase back to the point where the error was made, then correct the mistake and continue typing. Instead of a DELETE key, some terminals use the RUBOUT or BACKSPACE keys.

Use the RETURN key when you want to signal that you have finished typing a line. Usually commands or instructions that you type are executed when you type RETURN. Generally, you may change the command by means of DELETE, until you type RETURN.

Like the SHIFT key, the *Control* key, labeled CTRL, is held down while some other character is typed. Characters formed in this way are called *control characters*; they are used to extend the vocabulary used when talking with the computer. The following shorthand notation will refer to control characters in this manual. Rather than write "hold down CTRL and type C" or "type control C", "CTRL/C" (or sometimes "^C" or "↑C") will be used to refer to control characters. When a control character appears on your screen, it will usually be printed in the form "^C".

CTRL/C (control C) is the character that you type to initiate a terminal session. CTRL/C is also the interrupt character; type CTRL/C twice to stop most programs.

CTRL/Q allows another *screenful* of output to be shown. The computer can fill a display screen (usually 24 lines) in just

two seconds, which is faster than most people can read. Rather than deluge you with information, the computer pauses and beeps the terminal whenever it fills up the screen. Type CTRL/Q to signify that you want to see more output.

Warning:
If the computer is waiting for you to type CTRL/Q, ***nothing but*** CTRL/Q ***will make anything else appear on the screen. If the terminal won't do anything for you, try typing*** CTRL/Q ***before seeking further assistance.***

CTRL/U erases the line you are currently typing. Use CTRL/U instead of many repetitions of DELETE to erase the entire line. On some terminals the characters you delete will not be erased from the screen, but the cursor will move to the left side of the screen; you may then type over the old characters.

The CTRL/R character causes most programs to retype the input line. If, while you are typing, your terminal receives a message, you may type CTRL/R to re-view your current line.

Typing CTRL/O makes the computer stop typing. The computer will throw away terminal output. The computer will resume typing output either when you type a second CTRL/O or when it pauses for further input. This is useful if a program begins to type a large amount of material you don't want to read. You can abort the output without otherwise affecting the program.

The terminal key labeled ESC is called *escape*. This key is used for different purposes by different programs.

The space bar is the long unlabeled key along the edge of the keyboard nearest you. SPACE is used to separate words in forming commands and in many programming languages.

The terminal has special keys for the digits 1 and 0 (one and zero). These keys must not be confused with the keys for the letters lowercase L and uppercase O. Although these letters may look almost the same as the digits 1 and 0, you must use the digit keys when you are typing numbers.

We discuss terminals and the functions of various special keys in more detail in section 10.2, page 53.

There are many manufacturers of terminals and hence not all terminals are alike. Mostly they differ in the positioning of their keys, although they may have some internal incompatibilities that might make transferring between them difficult.

1.2. Starting a Terminal Session

Since there are three ways to hook a terminal to a computer, there are three somewhat different ways to start a terminal session. We will describe each of these ways.

1.2.1. Hardwired Terminals

Terminals that are hardwired to a Stanford DEC-20 display a distinctive banner when they are vacant. For example a hardwired terminal on LOTSA would bear this message:

```
Stanford LOTSA, TOPS-20 Monitor 5.3(5715)-4

Welcome.  To get started, hold down CTRL and type C.
```

To start a terminal session on such a terminal, you type CTRL/C. After a couple of seconds, the computer will respond with a message something like:

```
Stanford LOTSA, TOPS-20 Monitor 5.3(5715)-4
@
```

The message identifies the system and sometimes includes a forecast of system downtime or other messages of general interest. The computer also types the "@" (at-sign) character on the next line. This character is called a prompt; a prompt signifies that a program in the computer is awaiting your next command. The "@" character signifies that the EXEC program is waiting to serve you. The EXEC is the program which enables you to create files, run other programs,

log in, log out, and so forth.

If you use the LOTS computers, you often notice that the place is so busy that there are no vacant terminals for the computer you need. If this is the case, you must wait for a terminal to become available. There is a *queueing terminal* at which you may add your user name to the list of users waiting for a terminal; after the people in line ahead of you have been given terminals you will be assigned a terminal. When you are assigned a terminal, instead of the welcome message, there will be some message on the terminal identifying it as being assigned to you. When there are people waiting for terminals, the LOTS system will not permit you to log in on a terminal that is not assigned to you.

1.2.2. Dial-up Terminals

With a modem and a terminal it is generally possible to use the telephone system to connect to a computer. A modem is a device for converting between electrical signals that the terminal understands and sounds that can be transmitted over the phone system. There are usually two modems involved: the one at your end and the one at the computer's end. Different modems transfer information across phone lines at different rates, or baud. While hardwired terminals often run at 9600 baud, 300 and 1200 baud are common speeds for dial-up terminals. When getting ready to dial-up one of the DEC-20's, you should make sure your terminal and modem are both set to the correct baud rate with no parity, full-duplex transmission, and if necessary, either one or two stop bits. Check your terminal and/or modem instructions for how to set these parameters.

Dial-up numbers for a system are available by typing `HELP DIAL-IN` or `HELP DIAL-UP`[1] to the EXEC's prompt.

You will need to tell the computer what type of terminal you are using when connecting via phone lines. For information on how to set your terminal type, see Section 10.1. Note that when dialing in with any terminal that requires padding for carriage return and linefeed (e.g. Silent 700, Execuports, AJ630's, etc.), you should use type "TI".

The following is a general description of how to go about connecting your dial-up terminal to a DEC-20. The precise details will vary for different modem and terminal combinations.

- Turn on the terminal. Check the speed setting.
- If your terminal is labeled in `CPS` (characters per second), set it to either 30 or 120 `CPS`, depending on your modem. If your terminal is labeled in baud, set it to 300 or 1200 baud.
- Dial the correct computer telephone number; see `HELP DIAL-IN` for modem numbers or contact the facility staff.
- Wait for a steady tone or a high-pitched "beep" indicating the telephone connection to the computer has been made.
- If you have an acoustic coupler, place the phone receiver in the acoustic coupler. An acoustic coupler is a device to connect the telephone with a terminal if the terminal does not have a built-in telephone receptacle. If you are using a switchable modem (without a coupler), simply set the modem to "DATA" (sometimes abbreviated "DA") and hang up the phone. You will need to refer to your modem instructions for the exact procedure.
- Wait for the carrier detect light to come on. You will likely need to press the `RETURN` key or type `CTRL/C` to get the system banner and the EXEC's prompt, "`@`", to appear.

1.2.3. Ethernet Terminals

The Ethernet is a communications system that connects most of the major computer systems at Stanford. All of the Stanford DEC-20's are on the Ethernet. The Ethernet can be used for logging on to computers as well as for transferring data between computers. There are small special purpose computers on the Ethernet called "Ethertips" that have anywhere from 8 to 48 terminals on them. The Ethertip computer is responsible to connecting a terminal to a large

---

[1] Throughout this manual, DEC-20 "HELP files" will be referenced. The Stanford DEC-20's have a very easy-to-use system for accessing on-line HELP. For more information, see Section 6.4 in this manual.

general purpose computer such as a DEC-20 and supervising the exchange of characters between the terminal and the large computer. The term "host" is often used to refer to a computer on the Ethernet. From an Ethernet terminal it is generally possible to connect to any computer on campus.

A vacant Ethertip terminal will display a line near the bottom of its screen similar to the following example:

```
Type anything for an Ethertip Exec!
```

To start a terminal session, press any key. The Ethertip will respond with three or four lines of status information and then will prompt you with the Ethertip's name followed by an angle bracket. You then give the command "connect", a space, and the name of the computer to which you want to connect. After a few seconds the Ethertip will print out a confirming message and will connect you to the computer. Below is an example:

```
EtherTip 3.162, compiled Mon Aug 12 17:08:30 PDT on Navajo
Local host: "Tip-McCulloughA" Net#50 Host#16
Date/Time = 22-Sep-84 20:34:04, TIP booted on 19-Sep-84 15:03:23
escape char= ↑↑ local hold= ↑_
Tip-McCulloughA>connect sierra
is complete

Stanford Sierra, TOPS-20 Monitor 5.3(5714)-4
@
```

Sometimes the Ethertip will be unable to connect to the computer. This is because either the computer is not running (and hence can't respond to the Ethertip) or because a piece of the Ethernet communications network is temporarily malfunctioning.

1.3. Identifying Yourself to the System -- Logging In

You may now log in to the computer system. Logging in means telling the computer who you are. You must log in to gain access to your files and to use most programs. You identify yourself by the LOGIN command, your user name and your password.

An account on a DEC-20 permits you to use that computer and to store computer files. The password guards your account and files from use by anyone except yourself. You should not tell anyone else your password; your account is for your use only, and not for anyone else. The different DEC-20's have different criteria for account eligibility and the assigning of user names and passwords. If you have questions concerning accounts, please contact the staff of the relevant computer facility.

After you have successfully started a terminal session, the computer will prompt for the next command by typing the "@" character. When the "@" prompt is showing, type the word LOGIN, a space, your user name, another space, and your password. Then press RETURN. Since your password is supposed to be secret, it will not appear on the screen as you type it. What you type should be similar to the example below.

```
@login f.frank dingdong                    press RETURN after "dingdong".
```

When this appears on the screen the password, "dingdong", will not be visible. Each time you log in, the system will offer you the chance to read any new system messages. These are messages which are of general interest to most users. If this is the first time you have logged in to the computer, the computer will greet you with special attention, as shown below. If the computer responds that your account is "frozen" instead of letting you log in, you will need to contact the facility's staff to get your account activated. Below is an example of logging into the LOTSA computer for the first time.

```
@login f.frank
 Job 15 on TTY44  3-Aug-84 15:07
This is your first login.  Welcome to LOTS!  The following
are system messages (only some of which are relevant to you)
Type SPACE to see the message, DELETE to skip this message,
or "Q" to skip reading messages for now:

27 Jul 1984 1634-PDT  M.Mu  BMDP-79 Statistical Package
(435 chars; more?)
```

Franklin types DELETE to skip this message.

```
 3 Aug 1984 1451-PDT  Bob Knight <R.RMK>  Printer 0
(187 chars; more?) _
```

Franklin types a space to see this one.

```
Printer 0 (in CERAS 127) is down until tomorrow morning (unless the
serviceman gets here before then).  Sorry for the inconvenience.

Printer 2 (located on the right in CERAS 127), and Printer 1 (in
Terman) are operational, as are the LOTSB printers.

@
```

At the end of LOGIN and the messages, another prompt.

At this point you are logged in and the EXEC program awaits your next command.

1.4. Some Easy EXEC Commands

The "@" prompt signifies that a program called the EXEC is awaiting your command. The EXEC is a general purpose program that performs a wide variety of chores for you. It is the program that interprets the LOGIN and LOGOUT commands. The EXEC will run other programs for you, give you information about the status of the system and the condition of your programs and files, and will perform many other functions. Whenever the EXEC is finished doing one command for you, it prompts for another command by typing the "@" character.

When the EXEC runs some other program for you, that program may prompt for commands. In such a case, you must type commands for that program rather than typing EXEC commands. When a program that was started by the EXEC finishes running, it returns you to the EXEC; the EXEC prompts with "@". If you decide that you want to stop whatever program is running, you can type CTRL/C twice to make the EXEC stop the program and prompt for an EXEC command.

All EXEC commands start with a *keyword* that tells the EXEC what to do. Some commands require additional information. Each item of additional information in a command is called an *argument* (or a *switch*). The precise nature of these arguments (or switches) will be discussed in the sections on the particular commands. Every EXEC command is terminated by pressing RETURN. RETURN signifies that you are satisified that the command is correct and that you now want the computer to do it. The EXEC will be discussed at greater length in part II, starting on page 31. For the moment, we will illustrate only a few easily used EXEC commands.

Make sure the EXEC is prompting you with the "@" prompt. Type the word FINGER and press RETURN to see the names and locations of everyone using the system. Sometimes the list fills the screen. If this happens, just type CTRL/Q to tell the computer to continue typing information.

```
@finger
Username    Person            Job Jobnam Idle Line   Location
A.ANOTHER   Roc Blumenthal     32 DCALC        43 lobby center east #3
A.AYCHEN    ALICE CHEN         14 EDIT         26 Lobby wall carrel #2
D.DINKLE    Deborah Olander    44 EXEC        101 Terman NW Group, wall West
E.ERA       Patrick Kyllonen   20 SPSS      1   7 CERAS 507/LGI
F.FRANK     Franklin P. Bell   15 FINGER       44 Lobby center east #4
G.GRRR      grace mason        29 EDIT         25 Lobby wall carrel #3
J.JF        Jennifer Freyd     53 EMACS       171 Ethernet: Psych-VAX
J.JQJOHNS..J.Q. Johnson        34 MM        1  51 1200/150 dialin
L.LOUGHEED  Kirk Lougheed       7 SCRIBE      175 Ethernet: Sierra
M.MU        Sandy Lerner       41 DBEDIT       6 CERAS 124 Queenie's desk
O.OSPREY    Stephen Zagerman   54 PASSGO       61 Terman SW Group, wall West
Q.QLESS     Bonnie Warner      31 EDIT      2  32 lobby wall carrel #10
R.RMK       Bob Knight         42 SYSDPY      170 Ethernet: LOTS-B
SYSTEM      Not logged in      56 EXEC      4  15 lobby center west #3
W.WING      Ilan Kroo          49 EDIT         113 Terman NE Group, aisle East

@
```

Franklin Bell, who did this FINGER command, appears near the middle of the list. He is logged in as job 15 on terminal number 44, located in the center of the CERAS lobby.

To see the current date and time, type DAYTIME and press RETURN.

```
@daytime
 Tuesday, August 21, 1984 16:18:25
@
```

When Franklin asked for the daytime, it was 25 seconds after 4:18 in the afternoon (i.e., 16:18:25, using a 24-hour clock).

1.5. Changing your Password

One of the first things you should do after setting up your account is to change your password to ensure the security of your account. It prevents some other person from gaining access to your account and your files. To this end you should change your password frequently; setting a new password every six months is recommended. When selecting a new password, do not choose one that would be easy for another person to guess. Some suggestions on choice of password are:

- Choose a password at least five, and preferably six or seven, letters long. Use only letters and digits.
- Never use any part of your user name, your first or last name, the name of a friend, or the name of the computer, as your password. These are the first things someone who knows you would guess.
- Avoid common English words and proper names. Take the first letters of words in a phrase or run two words together. Examples of effective passwords are `tanstaafl` ("There Ain't No Such Thing As A Free Lunch") and `muchado` (changing a password is *not* "Much Ado About Nothing").

You may change your password by the EXEC command SET DIRECTORY PASSWORD. This command optionally takes a directory name (in pointed brackets) as its argument, defaulting to your own. After you type RETURN, the EXEC asks for the old password. Type the old password and RETURN. The password will not be displayed. Then the EXEC will ask for the new password twice. The reason you must type the password twice is to make sure that you typed it the same way both times. An example looks like this:

```
@set directory password
Old password: dingdong                    When you type these passwords,
New password: ringmychimes                they will not be displayed.
Retype new password: ringmychimes
@
```

1.6. Leaving the System -- Logging Out

If you just logged in and had to wait in line for a terminal, do not try this command now. If you do, your terminal may be assigned to someone else! The commands in the next seetions all require that you be logged in.

Just as you informed the computer of your presence by logging in, you must tell it you're leaving by logging out. It is very important that you log out when you leave because, if you don't, (1) no one else can use the terminal, (2) your allocation (at LOTS) or university billing account (on some of the other DEC-20's) will be charged for the time you waste this way, and (3) someone else might use your account and inadvertently damage your files.

To log out you must be at the "@" prompt. If you are not already at the "@" prompt, typing CTRL/C twice usually will get you there. Then you may log out by typing the command word LOGOUT and pressing RETURN.

```
@logout
Killed Job 15, User F.FRANK, TTY 44
  at 21-Aug-84 16:25:24, Used 0:1:24 in 1:17:45
```

The message about the time used means that the computer actually "worked" 1 minute and 24 seconds for F.FRANK during the one hour, 17 minutes and 45 seconds that he was logged in.

2. Files and What to Do With Them

Information such as commands, programs, or text can be saved in what are called *computer files*. You can think of a file on the computer as being similar to a file in an office file cabinet. You have control over the files which you create, and you get to decide who gets to use them. Instead of being in file folders with labels on them, these files are stored inside the computer. Your *directory* is your file-cabinet. Each file has a name and certain other information stored about it. You access a file by telling the computer the file's name. We will discuss these *file specifications* in more detail later on.

Among the operations that you can perform on files are:

- Make a new file where there was none before (`CREATE` command),
- Modify the contents of an existing file (`EDIT` command),
- Throw away the information in a file (`DELETE` command),
- Duplicate information from one file to another (`COPY` command), and
- Obtain information on what files you currently have (`DIRECTORY` command).

In order to use the computer effectively, you must learn how to do these manipulations.

There are many ways to store information in a file. Normally, if the information is text that must be entered by typing (i.e., information that you can read), you would use a special-purpose program called a text editor. There are several text editor programs on the DEC-20; the one called EDIT is recommended for beginners, and is described later in this manual. Using the EDIT program you can give a file a name and type in its contents.

Among the other ways to store information in a file is to bring information on a magnetic tape from some other computer. There are several programs on the DEC-20 for reading tapes. The TAPEIO program is especially suited for reading (or writing) tapes that come from (or go to) the Stanford I.T.S. IBM 3081 computer system (see page 235).

Files can also be transferred between computers on the campus Ethernet using the FTP program.

You may also run a program that writes a file itself.

2.1. File Specifications

Many kinds of information may be stored in computer files: the text of computer source programs, statistical data, etc. Since you may have many files, each file has a name, formally called the *file specification*, by which you may refer to the file. Usually you select a name that identifies the contents of the file. For instance, suppose you write a Pascal language program that calculates square roots. You might call the file containing this program `SQRT.PGO`, where `SQRT` is a common programming abbreviation for square root, and `PGO` is the file type denoting a Pascal language program using the load and go (or Passgo) compiler (the faster compiler).

A file specification is the *handle* by which you can grasp and manipulate the file and the information that it contains. A file specification may have several parts. One of the most common forms of file specification has two parts -- a *file name* and a *file type*, separated from each other by a period. Thus, in the file specification `SQRT.PGO`, "`SQRT`" is the file name and "`PGO`" is the file type. Both the name and type may be up to 39 characters long, but many programs on the DEC-20 are unable to handle names that are longer than six letters or types that are longer than three letters.

The file name is usually your free choice, selected to remind yourself of what the file contains. The choice of file type, however, is governed by convention. Usually, the file type indicates what kind of information the file contains. In the example above, "`PGO`" conventionally means a Pascal source program and indicates that you want to use the passgo compiler. Since the computer often uses the file type to determine what language the program is written in, it is a good idea to follow the established conventions.

Some of the common file types that are used on the DEC-20 appear below:

Type	Meaning
PGO	Pascal Load and Go (Passgo) program file
PAS	Pascal program file
FOR	FORTRAN program file
DAT	Program data file
SPS	SPSS command data file
HLP	Help system text file
TXT	Arbitrary text file
MSS	SCRIBE manuscript file
TEX	TeX manuscript file
DOC	Documentation text file
CTL	Batch control file
CMD	EXEC command file
LOG	Batch or PHOTO log file
REL	Relocatable Binary (object code)
EXE	Executable Binary (load module)
DVI	Device independent output file
PRESS	Output file for the Dover laserprinter

In other documents, there are somewhat different conventions about what to call the components of a file specification. In less formal usage the entire specification may be called the "file name" or just the "name". In this manual, *file name* means only one of the components of a file specification. However, "name of the file" may be used to mean "file specification", wherever the latter seems too repetitious. Also, the "file type" may be called the "extension" or "file extension" in other documents.

There will be more information about file specifications in section 9.1, page 35.

3. Creating a New File

Since you don't have any files yet, this is a good time to demonstrate how to make one. The example in section 3.1 shows the creation of a Pascal language source program. Even if you have no interest in Pascal, you should follow this example closely; in order to use the DEC-20, you will need to know how to enter and modify files.

Please read through page 24 before actually trying the following examples. To do these examples you will need to be logged in.

We create a file by means of the CREATE command. CREATE is an EXEC command, so you may use it only when the "@" prompt is showing. The CREATE command consists of the word CREATE, a space, the file specification of the file you wish to create, and RETURN. The file specification is simply the name you choose to call the file. In this example, the name chosen is SQRT.PGO. SQRT, an arbitrary choice, is mnemonic for "square root," which is one of the functions of this program. The file type, PGO, is chosen because this is a Pascal language program. The command you should use looks like:

```
@create sqrt.pgo
```

The CREATE command actually tells the EXEC to run a text editor program; a text editor allows you to make changes to a file without retyping the entire file. There are several different text editors available on the DEC-20. The two most common editors are called EDIT and EMACS; ZED and TVEDIT are two other popular editors. Section 3.1 will discuss the use of EDIT in creating a new file -- in this case, the text of a Pascal program. The program contains several errors, or *bugs*. Section 5, page 21 will demonstrate returning to EDIT to modify the file.

Rather than using EDIT, you may prefer to use another editor instead. For a discussion of EMACS, see chapter 20, page 135. Give the commands HELP ZED or HELP TVEDIT for details on how to use those editors.

3.1. Creating a File with EDIT

This section gives examples of only a few of the most basic features of EDIT. For a more detailed discussion of EDIT and its features see the pocket guide to EDIT, the *EDIT Users' Card* sold with this manual or available for a small charge at the LOTS office in the CERAS building. Also, see chapter 19, starting on page 97, of this manual for a more complete description of EDIT.

3.1.1. Typing the File into EDIT

When the EDIT program is started by means of the CREATE command, it assumes that you wish to create a new file, and allows you to type in the text you want.[2]

```
@create sqrt.pgo
Input: SQRT.PGO.1
00100
```

When EDIT starts, it tells you the name of the file that you are creating or editing. Note that the program types the file specification as "SQRT.PGO.1". The extra ".1" is the *generation number* of the file. File generation numbers usually start at one and count by one each time you change the file. Generation numbers are discussed in section 9.1.1, page 36.

Because you said CREATE, the EDIT program prompts us by typing "00100" which is EDIT's way of asking, "What do you want on line 100?" The EDIT program identifies lines by a five-digit line number and a page number. In the present discussion, we will generally ignore page numbers; the files shown in our examples will have only one page.

On each page in EDIT, lines appear in ascending sequence. Thus if you add lines to the file in the sequence 100, 200 and 150, they appear in the file in the order 100, 150, 200. When you change the file, the line numbers are handy reference

---

[2] If you already have a file called SQRT.PGO, the CREATE command will not allow you to destroy it. Choose another name for this example and retype the CREATE command using that name instead.

points that can be used when you talk to EDIT about particular lines. EDIT starts by numbering lines in multiples of 100. The spacing of 100 allows room for future insertions between the lines that are already in the file.

The correct reply to the line number prompt, "00100", is to type the text that you want placed on line 100. End the line, as usual, by typing RETURN.

```
@create sqrt.pgo
Input: SQRT.PGO.1
00100   PROGRAM Roots (OUTPUT);
00200
```

After you type RETURN, EDIT prompts with "00200" and waits for the text that you want to put on line 200.

As you type the text of the file, you may use the characters DELETE and CTRL/U to correct any errors you make, but these keys only correct mistakes that are on the current line. Once you type RETURN, DELETE and CTRL/U cannot be used to effect changes in that line. When you type this example, be sure that you have each line correct before typing RETURN. EDIT does have facilities by which you can correct mistakes on previous lines, but those will be discussed later.

We continue now by showing the remainder of the source program. (Spacing and capitalization are not important in this program.)

```
@create sqrt.pgo
Input: SQRT.PGO.1
00100   PROGRAM Roots (OUTPUT);
00200   VAR number: INTEGER;
00300       root: REAL;
00400   BEGIN
00500       FOR number := 0 TO 10 DO
00600       BEGIN
00700           root := SQRT(number);
00800           WRITELN (number, root)
00900       END
01000   END.
01100
```

The "END." in line 1000 signifies the end of the Pascal program. However, EDIT does not know that "END." means the end of the program, so it prompts us with another line number. To indicate that you are done typing the text of the program press the ESC (*escape*) key. EDIT responds with a dollar-sign, "$", to mark where ESC was typed and a star character, "*", on a new line.

```
@create sqrt.pgo
Input: SQRT.PGO.1
00100   PROGRAM Roots (OUTPUT);
00200   VAR number: INTEGER;
00300       root: REAL;
00400   BEGIN
00500       FOR number := 0 TO 10 DO
00600       BEGIN
00700           root := SQRT(number);
00800           WRITELN (number, root)
00900       END
01000   END.
01100   $ <--- Press the ESC key here, not the "dollar sign" key
*
```

3.1.2. The Print Command and Line-Ranges in EDIT

Just as the EXEC prompts for an EXEC command by typing "@", EDIT prompts for an EDIT command by typing "*". There are many EDIT commands, but you need only a few to start with. First, we will demonstrate the Print command.

The command letter "P" is used for the Print command, which displays parts of the file on the terminal. The command takes as an *argument* the line number of the line you want to see. Thus the command "P300" will make EDIT type line 300. EDIT commands are terminated by typing RETURN. Try it: type the command letter P, the line number 300, and press RETURN.

```
*p300
00300       root: REAL;
*
```

At the conclusion of the Print command, EDIT reprompts by typing another star.

To have a range of lines typed, type the command letter P, the first line number that you want to see, a colon, the last line number, and RETURN. The command "P200:500" prints all lines from line 200 through line 500.

```
*p200:500
00200   VAR number: INTEGER;
00300       root: REAL;
00400   BEGIN
00500       FOR number := 0 TO 10 DO
*
```

There are four special characters that you may use instead of typing a line number in an EDIT command. These characters are period, ".", caret, "^", star, "*", and "%". The character period, when used as a line number in an EDIT command, means the *current line.* The current line is usually the last line that was typed or changed by the most recent command. Since our most recent command was "P200:500", line 500 is now the current line. Try the command "P."; you will see it type line 500. The character "*" means the last line on the page. Try the command "P.:*" which should print lines 500 (the current line) through 1000 (the last line). The command "P*" will print line 1000.

```
*p.
00500       FOR number := 0 TO 10 DO
*p.:*
00500       FOR number := 0 TO 10 DO
00600       BEGIN
00700           root := SQRT(number);
00800           WRITELN (number, root)
00900       END
01000   END.
*p*
01000   END.
*
```

The caret character, "^" means the first line on the page. Try "P^:300".

```
*p^:300
00100   PROGRAM Roots (OUTPUT);
00200   VAR number: INTEGER;
00300       root: REAL;
*
```

The percent sign means the whole file. Try "P%". It will print all the lines in the file. The percent sign, however, should only be used for relatively short files. In general, you will want to use the other characters to print out only those lines in which you are interested.

```
*p%
00100   PROGRAM Roots (OUTPUT);
00200   VAR number: INTEGER;
00300       root: REAL;
00400   BEGIN
00500       FOR number := 0 TO 10 DO
00600       BEGIN
00700           root := SQRT(number);
00800           WRITELN (number, root)
00900       END
01000   END.
*
```

The various forms of the P command have demonstrated the concept of a *line-range* in EDIT. Things like "100:400" and ".:900" are line-ranges that specify a number of consecutive lines in the file. Many EDIT commands require a line number or a line-range as an argument. There are several other forms of line ranges that are also useful. A detailed description of the various forms of line ranges may be found in the *EDIT Users' Card.*

3.1.3. Leaving EDIT

The next command is extremely important. In the EDIT program, none of the changes you make to a file are permanent until you explicitly request that they be saved. One of the commands that saves the changes that you have made is the Exit command. The Exit command writes the changes you have made to the file, leaves the EDIT program, and returns you to the EXEC.

Type the command character E, signifying Exit, and press RETURN.

```
*e

[SQRT.PGO.1]
@
```

When EDIT exits, it types the name (including the generation number) of the file that it is writing the changes into, then returns control to the EXEC, which prompts for the next command by typing "@".

The changes you make to a file are not permanent and are vulnerable until you give the E command. There are some other commands, explained on page 23 and in the *EDIT Users' Card*, that also make your changes permanent.

4. Executing Your Program

You now have a file, SQRT.PGO, that contains the text of a program. The next logical thing to do is to run the program to see what it does.

To compile and run a program, you must issue the EXECUTE command. Type the word EXECUTE, a space, and the name of the file containing the program to translate. Type RETURN to terminate the command.

```
@execute sqrt.pgo
Stanford LOTS/Passgo 20 [SQUARE] -- 1..
Runtime:   0: 0. 94
[ROOTS  execution]
OUTPUT     :  <---  type the RETURN key here to indicate that output should go to the terminal
           0    0.000000E+00
           1    1.000000E+00
           2    1.414214E+00
           3    1.732051E+00
           4    2.000000E+00
           5    2.236068E+00
           6    2.449490E+00
           7    2.645751E+00
           8    2.828427E+00
           9    3.000000E+00
          10    3.162278E+00
@
```

The EXECUTE command must first decide what kind of program this is. The *file type* of the file containing the program indicates what kind of program it is. Since our program is in the Pascal language and since we want to use the fast Pascal compiler, we have given the file type PGO, which indicates that the Passgo compiler should be used. A table of the file types appropriate for source programs appears below:

Type	Meaning
FOR	FORTRAN compiler
SAI	SAIL compiler
PAS	Pascal compiler
PGO	Passgo (fast Pascal, no REL file)
BAS	Basic compiler
MAC	Macro assembler
FAI	Fail assembler

The results of the program execution above are the integers from zero to ten and their square roots (the E+00 is scientific notation. It indicates that each of these numbers is times ten to the 0th power, or 1). When the program finishes, it stops itself and returns control to the EXEC; the EXEC then prompts for another EXEC command.

If your results were significantly different from this example, you may have incorrectly typed in the source program. Steps to correct that kind of problem can be found in section 5, page 21.

If there are errors in the source file, the translator will type appropriate error messages. You should note the messages, correct the portions of the source file that are in error, and repeat the EXECUTE command. The correction process is demonstrated in section 5.

4.1. Stopping a Program

Sometimes you may discover that things are going wrong and wish to stop the program you are running. If you press CTRL/C twice the EXEC program regains control, stops whatever was running, and prompts for a command by typing "@". Since this emergency exit does not give the program a chance to finish, use it only as a last resort.

If you change your mind, you may be able to continue what you interrupted. Try typing the CONTINUE command to the "@" prompt.

Caution: avoid typing CTRL/C while in the EDIT program. If you interrupt EDIT via CTRL/C you may lose all changes made during the current editing session.

4.2. Checking the Status of a Program

If you are curious and want to know how your program is doing, you can check its progress even when it is in the middle of running. To do this, type CTRL/T.

The response from CTRL/T shows the time of day, the status of your program, the amount of computer time you have used, the length of time that you have been logged in, and the load on the system.

In the example below, CTRL/T is typed immediately after the computer typed "1..". This example is for your general guidance only; it may not work exactly as shown below when you try it.

```
@execute sqrt.pgo
Stanford LOTS/Passgo 20 [SQUARE] -- 1..
Runtime:   0: 0. 94
[ROOTS  execution]
OUTPUT     :  <---  Type CTRL/T here, then press the RETURN key.
  22:47:27 PASSGO IO wait at 534723  Used 0:01:36.9 in 1:34:47, Load  5.3
           0    0.000000E+00
           1    1.000000E+00
           2    1.414214E+00
           3    1.732051E+00
           4    2.000000E+00
           5    2.236068E+00
           6    2.449490E+00
           7    2.645751E+00
           8    2.828427E+00
           9    3.000000E+00
          10    3.162278E+00
@
```

The information that the computer types in response to CTRL/T may be interpreted as follows:

1. The time at which the CTRL/T was typed was 10:47 pm (22:47:27 on a 24-hour clock).

2. The Passgo compiler, which is called PASSGO, is in an I/O wait. It is waiting for you to input the file specification for where output should go. You want output to go to the terminal, so you hit the RETURN key. At the time CTRL/T was typed, it was executing the computer instruction located at memory location 534723. Some of the status messages you might see are:

   RUNNING — your program is currently running.

   IO wait — usually this means your program is waiting for you to type something.

   HALT — your program has stopped running. The EXEC awaits a command.

   ^C from Running — you stopped your running program by typing CTRL/C twice.

   ^C from IO wait — you stopped your program when it was awaiting input, by typing CTRL/C.

   There are some other status messages that CTRL/T may type, but they are less common.

3. The figures after the word "used" are the computer time (in this case one minute, thirty six point nine seconds) and console time (one hour, thirty four minutes and forty seven seconds) that you have used during this session.

4. The number after the word "load" is a *load average*. A load average is the number of people running

programs on the computer, averaged over a time period, in this case one minute. A load of 5.3 means that the computer is roughly 5 times slower than it would be if you were the only one on the system. Generally, a small number as the load average means that the computer will be able to give you prompt response.

If a program was waiting for you to type something, e.g., a command, when you typed CTRL/T, then it will still be waiting, although not prompting, after the EXEC has responded with the status information you requested. You may type CTRL/R to view the current prompt and current line again after the EXEC types the response to CTRL/T.

5. Changing a File

Usually, you will not be so fortunate as to complete a program in just one attempt. This section describes how to change the program that you have written. More detailed descriptions of the EDIT commands used to change files are given in the *EDIT Users' Card* and in the *LOTS EDIT Users' Guide.*

If you find you need to change the contents of an existing text file, you may use the EDIT program. You used the CREATE command to summon the EDIT program when there was no file. This time, however, a file already exists, so you must use the EDIT command to call the EDIT program. The command word EDIT is followed by the file specification of the file you wish to change. As with other EXEC commands, you must press RETURN to terminate the command.

```
@edit sqrt.pgo
Edit: SQRT.PGO.1
*
```

The EDIT command starts the EDIT program. The EDIT program types the file specification of the file you are editing, and prompts for an EDIT command.

Type the command "P%". As discussed in section 3.1.2, page 14, this command will type every line of the file, from the first to the last, on the terminal.

```
*p%
00100   PROGRAM Roots (OUTPUT);
00200   VAR number: INTEGER;
00300       root: REAL;
00400   BEGIN
00500       FOR number := 0 TO 10 DO
00600       BEGIN
00700           root := SQRT(number);
00800           WRITELN (number, root)
00900       END
01000   END.
*
```

You will change this program to include the squares of the numbers from zero to ten, in addition to their square roots. To do this you must first calculate the square, and then print it out. Since this requires one more Pascal-language variable, you must also include a declaration of that variable.

5.1. Insert a Line

You might begin by adding an entire new line to the file. Use the Insert command to add a new line. Type the command letter I, the line number at which to begin inserting, and RETURN. In the example below, the command is "I600". But note that the line number that EDIT offers is not 600, it is 650. Line 600 cannot be inserted because it already exists. Because the Insert command will not let you destroy an existing line, it offers a line midway between line 600 and the next line. Type the text of the new line and RETURN.

```
*i600
00650       square := number * number;
*
```

After you type the new line and RETURN, EDIT types "*" and awaits another command.[3]

5.2. Delete a Line

The variable square that you have just added to the Pascal program must be declared. You must change the line where the variable root is declared to include a declaration of square as well.

First, you will use the Delete command in EDIT to remove the old declaration on line 200. Then a new line, containing

[3]Sometimes, instead of returning to the "*" prompt, the insert command will continue prompting with line numbers. If this happens, you may type the ESC key to return to the command prompt, or you may continue inserting new lines.

both declarations, will be inserted.

To delete a line, type the command letter D, the line number, and RETURN.

```
*d200
1 Lines (00200/1) deleted
*
```

The Delete command removes the specified line from the file. Before prompting for another command, EDIT tells you which lines were deleted. The notation "00200/1" means line 200 on page 1.

Next insert the new declaration. Type the command "I." and RETURN. Remember that the period means the current line. The current line is 200, since that was the last line affected by the previous command. Thus, at this point, the command "I." is precisely equivalent to the command "I200". When EDIT prompts with the line number, type the new version of line 200 as shown. Type RETURN to end the new line.

```
*I.
00200   VAR number, square: INTEGER;
*
```

5.3. Replace a Line

Since it is cumbersome to type both a Delete command and an Insert command, there is an EDIT command that combines them both. The Replace command functions exactly as a Delete followed by an Insert.

You have one more line to change. You must include the variable square in the WRITELN statement on line 800. Use the Replace command to make this change. Type the command letter R, the line number 800, and RETURN. Then type the new text that you want on line 800.

```
*r800
00800           WRITELN (number, square, root)
1 lines (00800/1) deleted
*
```

As with the Delete command, EDIT tells you which lines have been deleted before prompting for the next command. These lines, of course, refer to the old lines, not to their replacements.[4]

It is now time to review the changes that you made. Use the command "P%" again to see the new version of the file.

```
*p%
00100   PROGRAM Roots (OUTPUT);
00200   VAR number, square: INTEGER;
00300       root: REAL;
00400   BEGIN
00500       FOR number := 0 TO 10 DO
00600       BEGIN
00650           square := number * number;
00700           root := SQRT(number);
00800           WRITELN (number, square, root)
00900       END
01000   END.
*
```

5.4. Leaving EDIT and Re-Running the Program

From the previous example you know that you may type the command letter E to exit and save these changes. You might then type the EXECUTE command to run the new program. Rather than leaving EDIT by means of the Exit command, you may prefer to try the Go command.

---

[4] As with Insert, the Replace command may prompt with more line numbers before returning to command level. If this happens, you may type the ESC key to return to the "*" prompt.

The Go command leaves EDIT and saves your file as though you had done an Exit command. Then, instead of returning to the EXEC for another command, the Go command redoes the previous EXECUTE command that you requested during this terminal session. Type the command letter G and RETURN. Note that when EDIT types the file specification of the file it is writing, the generation number will be incremented from one to two, e.g., SQRT.PGO.2.

Warning:

When EDIT writes the changes into a new generation of the file, the old generation is deleted. When a file is deleted, it means that the file is marked for removal at some later time. Because the file is not actually removed immediately, there are two things you should be aware of. First, if you blunder in EDIT, you may be able to recover an old version of the file if you have not already logged out. Second, the old versions of the file accumulate, and count against your disk space allocation. More information about both of these effects is given in section 9.5, page 43, and on page 27.

```
*q

[SQRT.PGO.2]


Stanford LOTS/Passgo 20 [ROOTS ] -- 1..
Runtime:   0: 0. 94
[ROOTS  execution]
OUTPUT     :
           0           0    0.000000E+00
           1           1    1.000000E+00
           2           4    1.414214E+00
           3           9    1.732051E+00
           4          16    2.000000E+00
           5          25    2.236068E+00
           6          36    2.449490E+00
           7          49    2.645751E+00
           8          64    2.828427E+00
           9          81    3.000000E+00
          10         100    3.162278E+00
@
```

5.5. More EDIT Commands

There are several more EDIT commands that deserve mention. All of the EDIT commands are explained in greater detail in the *EDIT Users' Card.*

5.5.1. Insert at the End of a File

Use the command "I*" to insert lines after the last line of the current page. Using the Insert command causes the EDIT program to enter *insert mode*, in which EDIT prompts with line numbers. Type the ESC key to go from insert mode to EDIT command mode.

5.5.2. Insert in the Middle of a File

Use the Insert command to insert lines in the middle of the file. Type the letter I and the line number after which you wish to begin inserting. If you want to insert more than one line at that point, type a comma and a line increment after the line number where you want to start. Thus the command "I500,10" means start at 500 (but, if line 500 already exists, start at 510) and insert lines 500, 510, 520, etc., through 590 (assuming line 600 exists). After you insert line 590, EDIT will revert to command mode. If you want to stop inserting before you get to line 590, use the ESC key.

5.5.3. Save the Current Changes

You may occasionally have to make a long series of changes to a file. Whenever you do so, there are two possible sources of damage to the work you are doing. First, you can damage the file yourself; for example, you might delete the wrong set of lines. Second, the changes you make to a file might be wiped out by the computer system *crashing.*

You can minimize such loss by means of the Backup command in EDIT. The Backup command writes the current state of your changes onto a file. The *backup file* contains all the changes that you have made so far. Type the command letter B and RETURN.

After using the Backup command you can regain all work that you did up to the point when you used this command. Each time you use the Backup command a new generation of the file containing all changes made thus far will be written. This new version will supersede the original file (or any previous backup version of the file).

Warning:

If you make a mistake and then use the Backup command, it will write out your file, mistakes and all, superseding the original copy of the file.

5.5.4. Abort the Current Editing Session

Sometimes you give EDIT the wrong command and damage the file that you are editing. If this happens, you can get out of EDIT without saving your (mistaken) changes in the file. The Quit command exits from EDIT without writing a file. Type the command letters EQ and type RETURN.

If you do save a mistaken set of changes, there is still hope of getting back your older version. Saving a new version of a file deletes the old version, but does not completely erase it from the disk. Until you log out or give an EXPUNGE command, you may be able to retrieve the older version by means of the UNDELETE command; see section 9.5, page 43, for details.

6. Where to Get Help

"HELP helps those who help themselves." -- The "HELP ME" command at LOTS

There are several potential sources for help on the DEC-20. By typing "?" and using the ESC key, you can often coax a program into telling you what command or part of a command you should type next. Many programs have some form of a HELP command that will print out internal documentation. There are also numerous documentation directories. And of course, you can always ask a person for help.

6.1. The Question Mark

The DEC-20 provides an extremely user-friendly environment. One hint which will carry you far in the process of learning how to use the computer is to remember to type a "?" when you're unsure what to type next. The "?" will provide you with a "menu" of available answers.

```
@?
```
Note: do not press RETURN *after the question mark*

will cause the EXEC to provide you with a compiete list of possible EXEC commands.

```
@information ?
```

will have the system tell you all the arguments (or subcommands) of the INFORMATION command. You may use the "?" at any point in an EXEC command line, that is, at any time in a command which is typed to the "@".

Although the above examples involve the EXEC, typing a "?" to most DEC-20 programs will give a list of possible responses. Notable exceptions include programs originally intended to run on a DEC-10 and most text editors. If typing a question mark doesn't work, try typing "HELP", "H", or "/H" and then pressing the RETURN key. It is extremely rare to come across a program that can't be coaxed into yielding some information on its use.

6.2. Using the ESC Key

Another extremely useful feature of the DEC-20 is its use of the ESC key to complete partially typed commands and to generate useful guide words in parentheses. Programs that will give you a menu when you type "?" will also respond to this use of the ESC key.

You need only type that portion of any command or file specification which uniquely distinguishes that command from any other, then press the ESC key. If you have not specified enough characters so that the command or file specification is ambiguous, the terminal will "beep" at you, and wait for you to type some more. If you are unclear about what you may type next, remember to use the "?" to get a "menu" of what commands are available, given what you have already typed.

Use the ESC key to get you through program commands and subcommands. A "?" may always be typed after ESC has completed a command, or a portion of a command for you. If you become familiar with the ESC key and "?", the DEC-20 will prove to be a quick and effective servant.

6.3. Example of Using "?" and the ESC Key

What follows is an example of figuring out on your own how to use the EXEC's COPY command to make a copy of a file. We will use the symbol "*<esc>*" to indicate that the ESC was pressed and "*<RETURN>*" to indicate that the RETURN key was pressed. The underlined text is what you would have typed in; everything else would be printed out by the computer.

First we type the first three letters of the COPY command, then press the ESC key. The EXEC finishes typing the word "copy", then prints out a guide word (also referred to as a noise word). Note that you never need to type a guide word yourself.

```
@cop<esc>Y (FROM)
```

To find out what comes next, we type a question mark.

```
@copY (FROM) ? FILE NAME
```

Apparently the COPY command wants the name of the file we are copying from, that is, the source file. So we start typing the file name, then press the ESC key. The DEC-20 finishes typing the filename for us and prints another guide word.

```
@copY (FROM) log<esc>TN.CMD.4 (TO)
```

We suspect that another filename is called for, the destination file, but we type another question mark to make sure.

```
@copY (FROM) logTN.CMD.4 (TO) ? FILE NAME
```

Sure enough, it wants the name of a file. We'll copy the file LOGIN.CMD to a new file, one that doesn't exist. Since the destination file doesn't exist, we can't use the question mark to complete the filename. After typing the file name, we type a space (to tell the DEC-20 that we're done typing the name), then we type another question mark to see what to do next.

```
@copY (FROM) logTN.CMD.4 (TO) test.cmd ? Confirm with carriage return or comma to enter subcommands
```

The phrase "confirm with carriage return" means that if we want to finish the command, we should press the RETURN key. If we didn't want the command to execute, then we should type CTRL/U or CTRL/C. The second part of the help message indicates that we should type a "," if we wanted to add subcommands to the COPY command. Subcommands modify the action of commands and can be very useful. In this case we don't want any subcommands, so we just press return. The DEC-20 then copies the files and returns us the EXEC prompt.

```
@copY (FROM) logTN.CMD.4 (TO) test.cmd <RETURN>
 LOGIN.CMD.4 => TEST.CMD.1 [OK]
@
```

6.4. The HELP Command

Another way of getting help is through the EXEC's HELP command, a command that types out system HELP files. HELP files are written by users to explain commands, languages, programs, and so forth. Typically, they explain what the command/program/language does, where it comes from, and where full documentation exists.

To use the HELP command, type the word "help", a space, the name of a topic, and then press RETURN. You may type "HELP ?" to get a list of the more important topics. There are many topics; to see them all type "HELP *" and then press RETURN. You may also try the command HELP with no topic name, i.e., type HELP and RETURN. This will give you general help and describe ways to use the HELP system effectively.

For example:

```
@help downtime
```

will print a file on your terminal describing how to find out when the computer is next scheduled to be down, i.e. unavailable for use.

Note that many of the HELP files on the system you are using may have been written for other DEC-20's. This means that occasionally the file will explain options or whole programs/commands not available on your computer, or will refer you to further documentation in someone's directory. Although efforts are made to minimize such misinformation, erroneous HELP files do slip in. If you find such an inappropriate HELP file or suspect that the information in a HELP file is wrong, contact the system staff so that the problem can be corrected.

7. Common Pitfalls and How to Climb Out

The following is a compendium of some of the problems people often run into using the DEC-20. It would be a good idea to read through this section; any of these could happen to you.

7.1. The Terminal Doesn't Work

- Is the terminal plugged in and powered on? The power switch for most terminals is usually located in the back of the terminal
- Is the data line attached? The data line carries the electronic signals to your terminal from the computer and usually connects in the back of the terminal.
- Try typing CTRL/Q. Remember that the DEC-20 will type 24 lines on your screen, then stop and beep once. Only typing CTRL/Q will make the computer continue to print on your screen.
- Some brands of terminals are sensitive to static and will lock up. See if you can find some form of a RESET key on your terminal. You may have to consult your terminal's user manual for help.
- If you are using a dial-up terminal, make sure that both your terminal and modem are set for the correct baud rate and that both are set up for fullduplex transmission.
- If you are using an Ethertip terminal and you can't get a Ethertip prompt even though everything else looks okay, it may be that the Ethertip computer has crashed.
- Are other terminals working? If not, the computer may be down.
- If all else fails, find a responsible person to look at the terminal. Hitting the terminal will only make matters worse, even if it does make you feel better in the short term.

7.2. Disk Full or Quota Exceeded

Files are stored on a device called a "disk". The amount of storage reserved on the disk for your directory and its files is referred to your disk space or disk quota. Your initial disk quota varies from system to system as does the procedure for obtaining more disk space. The message "Disk full or Quota Exceeded" indicates that the DEC-20 intercepted your (program's) attempt to create more files than you have disk space to store them.

In many cases, typing EXPUNGE followed by CONTINUE will fix your disk space problem and continue your program. If this is insufficient, you may have to delete some files or obtain a larger disk allocation.

First, however, you should know how to diagnose the reason why you are running out of disk space. The EXEC command INFORMATION DISK will tell you about your disk allocation and how much you are using.

```
@information disk
 26(32) Pages assigned, 21 in use, 5 deleted
 50 Working pages, 25 Permanent pages allowed
 PS:<F.FRANK> Over permanent storage allocation by 7 page(s).
 20686 Pages free, 131314 used on PS:
@
```

There is quite a lot of information supplied by this command. The "21 pages in use" is what the VDIRECTORY command would show (your normal -- not deleted -- files occupy 21 pages). The QDIRECTORY of deleted files would show that your deleted files occupy 5 pages. These add up to 26 which is the first number that is reported.

The number in parentheses, which may not always be present, is the amount of space the system actually has assigned for you. In this case, the number, 32, differs from the count of normal files, 26, because there are files presently being written

into this directory. This example indicates that six pages belonging to incompletely created files have been written so far. When these files have been written, there will be no differences between these two numbers, and the number in parentheses will not be reported.[5]

Fifty working pages means you are allowed a total of 50 pages at any time. The EXEC will send you a warning message when you log out and leave more than 25 pages (your *permanent* allocation) on the disk. The 32 pages presently assigned are 7 pages more than the permanent allocation of 25 pages.

In the example, the entire public disk has 20686 free pages, and 131314 pages that are in use for storing files, directories, and system overhead functions. The free space on the public structure (PS) must meet the needs of all the users on the system.

Your deleted but not-yet-expunged files count against your disk quota. Every time you edit a file and create a new generation, the old generation becomes a deleted file. Thus, your directory may rapidly fill up with deleted files; this may necessitate frequent expunging.

You should also be aware that under some circumstances EDIT requires temporary files that total twice the size of the file that you are editing. This means that you may not be able to change any file that is larger than one third the size of your disk allocation.

Sometimes quota exceeded problems are caused by running a program that is writing results to a disk file. Such a program could have an error that causes it to loop, continuously writing data. That would soon fill up your disk allocation. If you suspect that a program loop is the cause of your difficulties, stop your program, then run it again after redirecting the program output to the terminal so you can see what is happening. The technique for redirecting program output to the terminal varies depending on the programming language.

In managing your disk space, remember the DELETE and EXPUNGE commands. When selecting files to be deleted, keep in mind that a file you typed in should be among the last that you delete. In contrast, files that the computer writes for you that are relatively easy to reconstruct should be among the first that you delete. Examples of files that you should consider deleting are those of file type REL, PRESS, or IMP, or any file that is the output of a program that you can easily run again.

Many users find that they cannot do their assigned work with the disk space allotted to them. People who use the statistical packages or text formatters, for example, often generate output files that are too large to fit on their directories. On every Stanford DEC-20 there are several large publicly accessible directories called "scratch" directories. The command HELP SCRATCH will tell you how to use the scratch directories on your system.

Sometimes it is possible to send output files directly to a line printer by specifying output to be the logical device LPT:. In other words, if you have been directing output to a file RUN1.OUT and then issuing the PRINT command on RUN1.OUT, you can instead specify output to be LPT: wherever you previously specified it to be RUN1.OUT and the information is sent directly to the printer. SPSS, MINITAB, and BMDP are examples of programs whose output can be redirected in this manner.

---

[5] You can find out what files you are presently using by means of the EXEC command INFORMATION FILES. You can make an incompletely written file permanent by using the EXEC command CLOSE. You can cause your present core-image -- the program you are running -- and all of its incompletely written files to be thrown away by the EXEC RESET command.

Another source of discrepancies between the space reported by VDIRECTORY and the parenthesized number in INFORMATION DISK may be files that are so thoroughly protected that you cannot see them (see page 47). If you think this may be the cause of some difficulty, talk to a staff member.

7.3. Program is in a Loop

People who are learning to write programs often, if not always, make mistakes. One type of mistake is that for one reason or another the program will not stop running. This kind of error is called an *infinite loop.* If your program doesn't do anything, such as ask for more input or write further results, it may be in such a loop.

Type CTRL/T to observe the behavior of the program. If the response to CTRL/T indicates that the program is *running,* you may have a loop. Wait a while and type CTRL/T again. If the program status still indicates *running,* note whether the CPU time has increased much since the previous CTRL/T. See also the description of the output from CTRL/T, page 18.

Unless you have reason to believe your calculation will take longer, a program that uses more than two or three CPU seconds is probably in some erroneous loop. You can stop it by typing CTRL/C twice.

If CTRL/T indicates IO wait, your program is probably waiting for input from the terminal, rather than looping.

7.4. LNKNSA No Start Address

This message means that whatever compiler you used (often it is FORTRAN) thinks so little of your source program that it didn't bother to finish compiling it.

If you didn't get any other error messages, you should delete the REL file corresponding to the source file you named and repeat the EXECUTE command.

7.5. LNKUGS Undefined Global Symbol

This message, which is followed by a list of names, indicates that some subroutines or functions that you called cannot be located.

If you recognize the subroutine names, then perhaps the EXECUTE command that you gave did not include the name of the file (or library file) in which the subroutine is defined. Repeat the EXECUTE command, and be sure that you mention the names of all relevant files or library files.

If you do not recognize the subroutine name, then search the text of your program for the name. Perhaps you made a typing error. FORTRAN programs are especially susceptible to this kind of error.

7.6. What to do When You Wipe Out your File

First of all, don't get into this situation! Particular examples of things to avoid are:

- When running the PHOTO program, don't name your source file as the log file!
- Never confuse the name of your source file with the name of a program's output file. In SPSS don't mention your source file in a SAVE FILE statement. In Pascal, don't give your source file name in response to the OUTPUT : prompt. If you give an EXECUTE command specifying several source files, be sure to separate the file names by commas.
- Never, never give the EXEC command SAVE and name your source file!
- Don't delete your source file.
- After you accidentally delete your source file, don't EXPUNGE it. UNDELETE it instead.

When you know that you have lost your file, *do not log out! Do not* EXPUNGE. The file you want may still be there, and if so is retrievable with the UNDELETE command. If you can't figure out the UNDELETE command, try asking for help.

If, despite all the safeguards, you manage to delete and expunge a file you want, not all is lost. Each DEC-20 installation

periodically makes tapes of every user's recently changed files. Your file can usually be restored to a fairly recent condition. You should not count on this, however.

7.7. The Computer Crashed and I Lost My Editing Work

Occasionally (but hopefully *very* infrequently) a DEC-20 will "crash", that is, stop running and then have to be restarted. Crashes can be caused by a number of things; power failures, failure of electronic components, loss of air conditioning, and software errors are common reasons for crashes. If you are on a dial-up or hardwired terminal, you will get the message "%DECSYSTEM-20 NOT RUNNING"; on Ethertip terminals you will get the message "[Host seems to have crashed.]".

Sometimes the crash will turn out to be just a service interruption and it is possible to pick up your work where you left off. If you are on a hardwired line, you will get the message "[DECSYSTEM-20 continued]". Ethertips and dial-ups seldom get that message. When using one of the later types of terminals, you should run the FINGER program after reconnecting to the computer, but *before* logging in again. If FINGER's display shows a job of yours whose location is "Detached", then you can get that job back using the ATTACH command. If you don't see a job of yours, then the computer has truly crashed and you will have to log in again.

Below is an example of F.FRANK using the ATTACH command. Note that you don't have to log in again to use this command.

```
@finger f.frank                          Look for a "detached" job
F.FRANK  Franklin Bell      32 EXEC      Detached
@attach f.frank                          Type the username after the ATTACH command
Password:                                Type your password on the nextline
^C                                       A CTRL /C is often printed here
@                                        You are now talking to the EXEC
```

At this point F.FRANK could use the CONTINUE command to resume his program at the point the service interruption occurred. There are more options to the ATTACH command; give the EXEC command HELP ATTACH for further details.

If the computer has indeed crashed and if you were in a text editor at the time, then all changes you made to your file since you last wrote it out or saved it have been lost. Forever. You have to type your changes in again.

To avoid turning a misfortune into a catastrophe, you should get into the habit of occasionally saving the file while you are working on it. It is better to lose 15 minutes of typing than to lose several hours worth. You can do this by hand, that is, explicitly writing out the file every so often, or you can give a command to the text editor that will cause it to write out a backup file every few minutes or every few commands. See the chapters on EDIT and EMACS for details on how to use the various "auto-save" commands.

Part II
Using the EXEC Command Language

The EXEC program provides a powerful set of commands to allow you to examine and manipulate your computer environment. In this section of the manual, we will describe some of the EXEC commands in greater detail.

In the following material, the most important information is contained on page 33 through page 46. These sections deal with EXEC commands in general and especially those for manipulating files. Although the remainder of the material is useful to some people, you might omit reading it until you are more familiar with other facets of the computer system.

We have talked about the EXEC already in *Getting Started on the DEC-20*. You should know by now that the EXEC is the program that *prompts* for commands by typing the "`@`" (*at-sign*) character.

All EXEC commands begin with a command word that specifies the nature of the command itself. Some commands require additional information following the command word. For example, the `LOGIN` command that we have seen requires that you type your user name after the word `LOGIN`. Additional items of information that a command requires are called *arguments*. All EXEC commands become effective when you type `RETURN` at the end of the command line.

8. Abbreviation and Completion of EXEC Commands

In this manual we try to spell out the entire name of each command we use. This is because the command names are usually significant, i.e., they are descriptive of the function they perform.

Once you have mastered an EXEC command, there is no need for you to spell the whole command name. We will see some rather long commands, such as `VDIRECTORY` and `INFORMATION OUTPUT-REQUESTS`. You need type in only that portion of the command that makes the command name unambiguous. For example, at present, `V` is sufficient for `VDIRECTORY` and `I O` suffices for `INFORMATION OUTPUT-REQUESTS`.

Another reason we avoid abbreviations in this manual is that the minimum abbreviation is subject to change as the EXEC program develops. If another command were added that begins with the letter "I" some more of the word "information", perhaps "in" or "inf" would be needed.

Generally, however, you may abbreviate EXEC commands to about three letters.

8.1. Command Completion

Another feature of the EXEC program is that it performs *completion.* After you have typed enough letters to specify a command, you may type the `ESC` key. If what you typed is an unambiguous command, the EXEC will complete the command name. When the EXEC completes a command name, it may type a *noise word,* a phrase enclosed in parentheses that prompts for the next field of the command.

If the portion of the command that you typed is ambiguous when you type `ESC`, the EXEC will beep (bell) the terminal and wait for you to type more letters. If no command matches the letters you have typed so far, the EXEC will tell you so and make you start over.

For example if you type the command "`ED`" and press the `ESC` key, the EXEC program will complete the command "`EDIT`" and supply the noise word "`(FILE)`" to signify that a file specification is needed.

8.2. Specific Prompting

The additional information that a command requires, such as the file specification in the EDIT command, is called an *argument.* Some commands take no arguments, some may take optional arguments, and others require arguments. Besides prompting with noise words, the EXEC will also tell you what argument is expected at any point. To find out what the EXEC wants to see next, type a "`?`" character.

For example, if you type "`E?`" to the EXEC it will reply with a list of commands that begin with the letter E. This list includes commands such as `EXAMINE`, `ECHO`, `EXECUTE`, `EDIT`, and `EXPUNGE`. Having offered a list of advice, the EXEC will retype the letter `E` and allow you to continue typing the command. If you type "`EDIT ?`" the EXEC will mention a list of possible `EDIT` command switches and say that a switch, or file specification, or carriage return is appropriate.

8.3. Indirection in EXEC Commands

Occasionally a situation will arise where you are repeating a long sequence of command arguments several times, or perhaps repeating the same long EXEC command. You can store a command line or piece of a command line in a file, then tell the EXEC to use the contents of the file to complete your command. Create a file with a type of `CMD` containing the command arguments. Then instead of typing the arguments to the EXEC command, just type an at-sign ("`@`") followed by the file name.

For example, suppose you have a file called `MAILLIST.CMD` containing a list of people to whom you regularly send mail. Then instead of typing this list each time you want to mail to all your friends, you can just type the file name:

```
@type maillist.cmd
00100   j.jqjohnson,g.gorin,q.queenie,r.rmk,b.bombadil
@mail @maillist.cmd
Subject:
```

These features generally make dealing with the EXEC program easier. If you forget the sequence of command arguments or the form of a command, command completion and prompting are quite helpful tools to refresh your memory. Command abbreviation and indirection save a tremendous amount of unnecessary typing.

Another type of command file, called a "MIC" file, is also useful for handling long sequences of commands. See Chapter 24, page 161 for a description of how to use the MIC facility.

8.4. Recognition of File Specifications

Just as the EXEC (and some other programs) will complete the name of a command when you type the ESC key, many programs on the DEC-20 will complete the name of a file when you type part of the name and press ESC. This process is known as *file-name recognition* (even though it is used to recognize an entire file specification, not just the part we call the "file-name").

Different programs do different things when you attempt file-name recognition. If a program is looking for the name of an existing file, you must supply enough of the specification to identify one file. When you type part of the file specification and press ESC, the computer will complete the file specification for you if the fragment you have typed uniquely identifies one file. Otherwise, if the file name you have identified corresponds to several files with different types, the computer will finish typing the name, type a period, and beep. You may then specify as much of the file type as necessary to make it unique, and type ESC again.

If a program expects the name of a new file, you will generally have to type the entire file specification.

9. EXEC Commands to Manipulate Files

The EDIT program allows you to access and change individual lines and characters within a file. In contrast to the EDIT program, the EXEC commands manipulate entire files. The EXEC has commands to perform such file manipulations as:

- Display the name, size, and date of each of your files,
- Duplicate files,
- Show the contents of a file on your terminal,
- Print the contents of a file on the printer,
- Delete files,
- Change the names of files, and
- Define the access protection of files.

Before we can discuss the EXEC commands for manipulating files, it is necessary to explain a few more ideas about files in general.

Every file has an owner. The files you make naturally belong to you. All of the files that you own are collected into your *file directory* which is simply the place where each of your files is kept.

Files are physically stored on a magnetic disk. The computer can read and change the information stored on the disk in much the same way that a home tape recorder can save and play back sounds. All users share the space on the disk for their file storage.

Each directory is limited in capacity; there is a maximum total size of files that you cannot exceed. This limit is called your disk quota.

9.1. File Specifications

File specifications have more parts than we have yet discussed. A more complete specification of the file we have been calling SQRT.PGO is PS:<F.FRANK>SQRT.PGO.2. Franklin Bell's user name is F.FRANK; his file directory is called <F.FRANK>. All of Franklin's files reside in his file directory, so the complete formal specification for each of his files includes the directory name, <F.FRANK>. Finally, the "PS:" portion of the file specification is the *device name* of the physical disk where Franklin's files (and his file directory) reside. "PS" means *Public Structure*, which is where most people's files reside.

You may think of a file specification as being the address that describes where a file is kept. It is quite similar to the address that you would put on a letter, except it is written with the largest geographical area first. It is as though the Post Office would accept a letter addressed as:

California
Palo Alto
Stanford University
CERAS Building
Room 124
LOTS Computer Facility
Manager

Where each successive piece of address gets you closer to the intended destination. In this light, the device name "PS:" specifies which physical piece of computer hardware Franklin's files reside upon. Then the directory name <F.FRANK> specifies his file directory, in contrast to the several thousand other file directories. The file name, SQRT, selects one of the several files in his directory. The file type and generation PGO and 2 further specify a particular file, until there is no room for ambiguity, and no possibility of reaching the wrong address.

Capitalization is unimportant in file specifications. However, punctuation is crucial. Special characters such as "<" and ">" separate various parts of the file specification. Note, though, that spaces may not occur anywhere in the file

specification. For example,

```
@type ps:<f.frank>sqrt.pgo
```

is a legal file specification, but

```
@type ps: <f.frank> sqrt.pgo
```

is not.

9.1.1. Defaults in File Specifications

As we have demonstrated in previous examples, it is not necessary to specify the entire file specification when you talk about a file. Continuing the analogy above, if you were having coffee at the Stanford Coffee House, and were asked how to find the LOTS office, you would probably omit mention of California, Palo Alto, and Stanford University when you were giving directions. Similarly, when Franklin Bell is logged in, he omits mentioning `PS:` and `<F.FRANK>` when he talks about his files, though he always needs to mention the file name (and often needs to mention the file type).

When Franklin finds it necessary to refer to files belonging to someone else, he uses that other person's directory name to refer to those files. Also, if Franklin had to refer to files on some computer peripheral other than the public structure, then he would mention the device name in the file specification.

When the directory name is omitted from a file specification, the name of the *connected directory* is used by default. Usually, your connected directory is the directory where you are logged in, but it may be changed. See page 37.

If you omit mentioning the generation number in a file specification, the computer does the most intelligent thing. In most programs, when you read a file you get the most recent generation. Usually, when you supersede an existing file the generation number of the new file is one higher than that of the old one.

The generation number `0` indicates the highest existing generation number (most of the time this will be the only existing generation number). An asterisk (`"*"`) indicates all generations, `-1` indicates the next highest and `-2` indicates the lowest.

9.1.2. Wild-Cards in File Specifications

In some commands that accept a file specification as an argument, you may specify a asterisk (`"*"`) in one or more fields of the file specification. The `"*"` in a file specification is a *wild-card*. The `"*"` matches anything in the position where it appears.

For example, in the `DIRECTORY` command (see page 42), you might type `"*.PGO"` as the file specification, in which case you would obtain the directory listing of all your files of type `"PGO"`, no matter what name they had.

Another type of wild-card is the percent character, `"%"`. A `"%"` in a file specification matches any single character. Thus, you could write a file specification such as `F%X.*` which would match files named `FOX.PGO` or `FIX.FOR`, but not `FX.PGO` or `FXY.REL`.

You may also use the `"*"` character to mean any string of zero or more characters. The file specification `FORTRAN*.*` could match file names such as `FORTRAN.GUIDE` or `FORTRAN-HELP.LOG`, or any name that begins with `"FORTRAN"`.

9.1.3. Logical Names

There are several file directories that people frequently refer to. To make referencing these directories easier, the computer has some special names, called "logical names", that really refer to these directories. For example, files that contain documentation about programs reside in the directory `PS:<DOCUMENTATION>`. Because these documentation files are referenced so often, we have a shorthand notation, `"DOC:"`, meaning `"PS:<DOCUMENTATION>"`. This shorthand, `"DOC:"`, is a logical device name. More precisely, a logical name supplies default values for portions of the file

specification that follows it.

Similarly, the *help files* (see page 26) reside on HLP:, and the editor that the EDIT command invokes is on EDITOR:. There are other logical names defined in a similar fashion. For a list of system logical names, use the EXEC command INFORMATION LOGICAL-NAMES SYSTEM (which is currently abbreviated as "I L S").

You may define your own logical names by means of the DEFINE command. You can examine the definitions you have in effect by the command INFORMATION LOGICAL-NAMES JOB (usually abbreviated I L J).

As an example of the DEFINE command, note that in DEC-20 Fortran input-output unit 6 is normally the terminal unless the logical name "6:" is defined. To redirect output on Fortran unit 6 to the printer (device LPT:) use the command:

```
@define 6: lpt:
```

To cancel the effect of this definition use the command:

```
@define 6:
```

As another example, if you frequently refer the files in the directory PS:<S.STATISTICS> you may want to define a logical name such as S: to abbreviate this directory name.

```
@define s: ps:<s.statistics>
```

As we mentioned, logical names really just provide default values for otherwise incomplete file specifications, though usually they are used only to supply a default device or directory. They can even specify several alternative defaults separated by commas. If you have a logical name DATA: defined as <C.CS10X>,<S.STATISTICS> and give the command:

```
@define data:<c.cs10x>,<s.statistics>
@type data:drugs.dat
```

The system will look first for <C.CS10X>DRUGS.DAT but if that file is not found, the system will try to find <S.STATISTICS>DRUGS.DAT for you.[6]

Logical names you have defined take precedence over system logical names. Thus, you could change the editor that the EDIT command invokes by redefining EDITOR: as the name of an executable file containing your preferred editing program.

```
@define editor: sys:emacs.exe
```

9.1.4. Changing the Default Directory - the CONNECT command

When we said that Franklin Bell may omit the mention of his directory name, <F.FRANK>, when talking about his own files, we were not completely accurate.

When you log in, you are *connected* to your own directory. When you omit mention of the directory in a file specification, the name of your connected directory is assumed.

It is possible to connect to another directory. Having done so, all file specifications that you give will default to that directory instead of your own. The EXEC command CONNECT takes as an argument a directory name (including pointed brackets) or a logical name which stands for the directory name (see page 36). Unless the directory has been set up for special access, you will be prompted for a password when you attempt to connect.

```
@connect ps:<r.ruth>
Password:              <-- Type the password, then press RETURN
@
```

[6] The alternatives in a logical name will only be searched if your program explicitly asks for an existing file. If you try to create a new file using a logical name, specification for the new file will always be built using the first alternative.

After giving the CONNECT command, all file references will default to the connected directory. If you wish to refer to files in your own directory, you must give their full names, including your device and directory.

To revert to the normal situation of being connected to the same directory as the one where you are logged in, type the command CONNECT and press the RETURN key.

A number of the Stanford DEC-20's have *scratch directories* for use when you are creating large temporary files that won't fit on your own directory. Anyone can connect to a scratch directory without a password. These directories have names of the form PS:<SCRATCH1>, PS:<SCRATCH2>, or PS:<1SCRATCH>, PS:<2SCRATCH>, and so forth. There are often logical names such as SCR1: and SCR2: to refer to scratch directories. Give the command HELP SCRATCH to find out the precise names of the scratch directories on the DEC-20 you are using.

Warning:

Scratch directories are temporary and not secure. Anyone may read or delete files on them, and files are automatically deleted within a few days. When you are done using a scratch directory, delete any unneeded files and move the rest back to your regular directory.

When connecting to the scratch directory it is a good idea to give your file an original name so that you don't overwrite someone else's file. Below is an example of connecting to a scratch directory (using the logical name SCR1:) and copying a file from a login directory to the scratch directory.

```
@directory *.pgo

   PS:<R.RUTH>
 SQRT.PGO.1

 Total of 9 pages in 1 file
@copy sqrt.pgo scr1:ruth-sqrt.pgo
 SQRT.PGO.1 => <SCRATCH1>RUTH-SQRT.PGO.1 [OK]
@connect scr:
@vdirectory *.pgo

   PS:<SCRATCH1>
 SQRT.PGO.1;P777777        6 12824(7)   18-Aug-84 13:56:45 F.FRANK
 RUTH-SQRT.PGO.1;P777777   9 22280(7)   18-Aug-84 14:06:45 R.RUTH

 Total of 18 pages in 2 files
```

To revert to the normal situation of being connected to your login directory, type the command CONNECT and then press RETURN. You should make sure to copy your file back to your login directory.

```
@connect
@copy scr1:ruth-sqrt.pgo sqrt.pgo
 <SCRATCH1>RUTH-SQRT.PGO.1 => SQRT.PGO.2 [OK]
```

For more information on use of the COPY and RENAME commands, see page 46.

9.1.5. Disk Structures

As we have remarked (page 35), one part of a complete file specification is the device name. Most people's files reside on the "public structure," or PS: device, but many system files are kept on other structures. When accessing a file on a device other than PS: you must specify the device name explicitly.

9.2. Viewing a File on the Terminal

The TYPE command directs the EXEC to type the contents of a file on the terminal. To the "@" prompt, type the command word TYPE, a space, the file specification of the file you wish to view, and RETURN. As our example, we show how to view the file SQRT.PGO.

```
@type sqrt.pgo
00100   PROGRAM Roots (OUTPUT);
00200   VAR number, square: INTEGER;
00300       root: REAL;
00400   BEGIN
00500       FOR number := 0 TO 10 DO
00600       BEGIN
00650           square := number * number;
00700           root := SQRT(number);
00800           WRITELN (number, square, root)
00900       END
01000   END.
@
```

Thus far, we have talked about text files only. A text file is a file that contains characters. Usually text files are created by someone typing. There are numerous other kinds of files besides text files. If a file is not a text file, it will not make any sense when you ask the EXEC to type it. Among the files that specifically are not text files and which are not suitable for the TYPE command are all files whose file type is REL or EXE.

9.3. Obtaining Printed Copies of Files

The EXEC command PRINT directs the computer to print a file on the line printer. Type the word PRINT, a space, a file specification, and RETURN. Below is an example of giving a PRINT command on one of the LOTS computers.

```
@print sqrt.pgo
[Job SQRT Queued, Request-ID 756, Limit 9, Node CERAS]
@
[From SYSTEM:  Job SQRT Request #756 Started Printing at 10:31:41]

[From SYSTEM:  Job SQRT Request #756 Finished Printing at 10:31:46]
```

The PRINT command responds with a message indicating that your command has been accepted and tells you some things about the request it has entered: "SQRT" is the name, and 756 is the sequence number of this request; the printer assumes a page limit of 9 pages. The page limits are based on the size of the file; page limits are usually more than sufficient to print the file. "Node" CERAS means the file is being printed on one of the CERAS printers.

When the PRINT command has completely entered your print request, it returns to the "@" prompt. At this point, you may continue your work, leave the system, or do whatever else you want, subject to some cautions stated below. There is no need to wait while your file is being printed.

In the example above, we haven't done anything else after the PRINT command. When the printer actually starts printing our file, it sends us a message. When it finishes printing, it sends us another message telling us that the listing has been completed. Please pick up your listings promptly.

Now, you ought to observe a strange thing about the example shown above. There is no prompt showing on the bottom line. What has happened is this: by means of your terminal keyboard, you are in a dialog with the computer. When you type a command, the computer responds, and you may then type another command. The messages about your printed output are not part of this dialog. That is, if the computer was waiting for some command from you, it is still waiting for that command, irrespective of any other messages that appear on the screen. Logically, therefore, we are still at the "@" prompt, even though the screen has changed. We can continue with our work as though the message had not appeared.

Although such *asynchronous messages* (i.e., messages that are not part of the dialog) are often useful and informative, they are also somewhat confusing. You can usually recognize such messages; often they appear between square brackets, "[" and "]". Also, in most cases, if you type CTRL/R you will get back the current prompt and any typing that you may have done on that command line (In some cases, CTRL/R doesn't help completely; even so, you can usually figure out what is happening.). For further discussion of asynchronous messages see section 12, page 57.

If we type CTRL/R now, the computer retypes the prompt character.

```
@print sqrt.pgo
[Job SQRT Queued, Request-ID 756, Limit 9, Mode CERAS]
@
[From SYSTEM:  Job SQRT Request #756 Started Printing at 10:31:41]

[From SYSTEM:  Job SQRT Request #756 Finished Printing at 10:31:46]
<--- Type CTRL/R here. The "@" prompt will reappear.
@
```

9.3.1. Cautions About the PRINT Command

The PRINT command directs the computer to print the specific file that you identified. If you change the file via EDIT before the computer prints the file, the new file will have a different generation number than the old one. Since the PRINT command remembers the old generation number (even if you didn't mention a generation number), when the computer gets around to printing your file it will not be able to find it. So, *do not change your file until after it is printed.*

Do NOT attempt to print anything but a text file. If you are in doubt, try the TYPE command first. If the results of TYPE are unintelligible, the results of PRINT will be far worse! Never PRINT any file that has file type REL, EXE, PRESS, DVI, IMP, MUSE, or BIN. Plot files, which contain information that an intelligent printer can translate to line drawings, are the only exception to this rule. Although a plot file is not text, it does make sense when printed on a *plotting* line printer, such as a Printronix printer.

Do not print multiple copies of the same file. A line printer is not a copying machine; you will annoy and inconvenience many people if you use it as one. Do not print reams of documentation; it is annoying and inconvenient to other people, and printed documentation is usually already available if you need your own copy.

9.3.2. Switches in the PRINT Command

If you are using one of the LOTS computers, you may force your output to go to either Terman or CERAS. There are four printers located in room 127 of the CERAS building and two printers located in room 104 of the Terman Engineering building. You may select the location for your printed output by means of a *switch* on the print command. A switch is a command modifier. Switches generally are typed in the same line as the command. A switch begins with the character slash, "/", and contains a keyword. To force your listing to the Terman printer, type "/DESTINATION-NODE:TERMAN" following the file specification in the PRINT command. Similarly, the switch "/DESTINATION-NODE:CERAS" forces output to come to one of the printers at the CERAS building. Incidentally, to find out what default destination the system will use for your jobs, you can give the command "INFORMATION JOB". If you are at CERAS and want all your listings from now on to default to Terman, you can give the command "SET LOCATION TERMAN".

Other computer facilities at Stanford have different ways of routing print requests to printers. Some, such as the Score computer at Computer Science, do not have a line printer. Consult the help files on your system to find out where the line printer (if any) is located and to choose a particular printer if there is more than one.

There are other switches to the PRINT command. For example, you may specify that your output not be printed until after some specific time. If you had a long listing you might delay it until very late at night as a courtesy to other users. The switch "/AFTER" allows you to delay your listing. Type a colon, ":", following the word AFTER, and a date and time; e.g., "/AFTER:20-Sep-84 0300" will delay a print request until 3 AM on September 20, 1984. Consult the HELP file on the PRINT command for details on other switches.

9.3.3. Examining the Status of Print Requests

Often many people are waiting for printed output. You may examine the status of your print request and the status of the print queue by the EXEC command INFORMATION OUTPUT-REQUESTS. Presently this command may be abbreviated as "I O".

```
@information output-requests

Printer Queue:
Job Name   Req#  Limit            User
--------   ----  -----  ------------------------
*  SQRT     756      9  F.FRANK                    On Unit:0
   Started at 10:31:41, printed 0 of 9 pages
*  FUGUE    528     36  P.PAD                      On Unit:1
   Started at 10:31:28, printed 8 of 36 pages
   LOGIN      7      9  J.JQJOHNSON                /Dest:TERMAN
            /After:31-Dec-84 10:40
There are 3 Jobs in the Queue (2 in Progress)

@
```

This command shows each job in the printer queue, the user name of the person who requested each listing, the location of the printer where the listing will be made, and any special switches that were specified in the print request. In the example there are jobs currently being printed on Unit 0 (one of the LOTS printers in CERAS) and Unit 1 (a LOTS printer in Terman). One other job is waiting to be printed in Terman. To see only the requests waiting to be printed at Terman, a command such as "I 0 /DESTINATION-NODE:TERMAN" could be given.

9.3.4. Canceling a PRINT Request

If you have requested a listing of a file and subsequently decide that you no longer want it, please cancel the request. You may cancel a print request by means of the CANCEL command. Type the word CANCEL, a space, the word PRINT, another space, the name of the print request, and RETURN.

The name of the print request can be determined from the INFORMATION OUTPUT-REQUESTS command. Usually the name of the print request is the first six or fewer letters of the file name that is being printed, or the name of the program that requested printed output.

For example, CANCEL PRINT SQRT would cancel the listing requested in the example above.

```
@cancel print sqrt
[1 Job Canceled]
@
```

If your request was being printed when you canceled it, the system would have added the phrase "1 was in progress" after telling you one job was canceled.

The CANCEL command distinguishes between things that are actually being printed when canceled, and requests that are merely in the printer queue. If a request that is being printed is canceled, it reports that as a job that was *canceled.* Requests that are queued but which have not yet started printing are reported as *killed.*

The CANCEL command will cancel all requests that match the name you specify. If you need to cancel only one of several requests that have the same name, you may use the "Request ID" number that appears in the INFORMATION OUTPUT display instead of the name. Specify the request number of the particular request you want to cancel, e.g., CANCEL PRINT 756. To cancel all your print requests, use the command CANCEL PRINT *.

When the printing is finished, a message will be sent to your terminal. As with the message about printing having started, you may type CTRL/R to view the current prompt and input line.

9.3.5. Modifying a PRINT Request

If you have requested a listing and then decided that you would like to change or add some switches, you can use the MODIFY PRINT command. To use this command, type MODIFY PRINT followed by the name of the print request, then the switches you want to add or modify. Each switch is preceded by a slash (/). To see a list of the possible switches type MODIFY PRINT/?.

Suppose, for example, you were using the LOTSA computer and you had requested a listing of the file HMAN.DAT from a

terminal at Terman. Using the I O command you see that the the Terman print queue is very long whereas the CERAS print queue is relatively short. You could send your listing to CERAS by typing the EXEC command

```
@modify print hman/destination-node:ceras
```

9.4. Obtaining Information about Files

A variety of commands exist that allow you to view the names of each of your files. These commands help you manage the disk space that you are allowed.

Two commands that allow you to view the names of your files are DIRECTORY and VDIRECTORY. The DIRECTORY command shows a list of all the files you have. The VDIRECTORY and other directory commands show more detailed information about each file.

9.4.1. Directory Command

The following is an example of the DIRECTORY command. Our friend, Franklin Bell, has been hard at work, so his directory has several files now.

```
@directory

   PS:<F.FRANK>
 BIGGER.FOR.2
   .REL.2
 DIR..3
 EDIT..2
 FINAL..1
 OPER.PAS.1
   .REL.4
 OUTPUT..3
 ROOT.FOR.1
 SQRT.PGO.2
 THIRD.PART.2

 Total of 35 pages in 12 files
@
```

The DIRECTORY command lists the (partial) file specification of each file, alphabetically by file name. If two or more files have the same file name, then the file name is listed only once; the different file types are then listed on successive lines. For example, note how the file BIGGER.REL.2 is listed. The entry immediately following BIGGER.FOR.2 is for the file BIGGER.REL.2, but the file name, BIGGER, has been omitted.

9.4.2. Verbose Directory Command

As shown in the example below, the verbose (wordy) form of the DIRECTORY command, VDIRECTORY, provides more information about your files.

```
@vdirectory

   PS:<F.FRANK>
 BIGGER.FOR.2;P777752      1 82(36)      4-Aug-84 08:10:50 F.FRANK
   .REL.2;P777752          1 1(36)       4-Aug-84 16:56:52 F.FRANK
 DIR..3;P777752            1 545(7)      4-Aug-84 10:29:10 F.FRANK
 EDIT..2;P777752           4 8273(7)     7-Aug-84 23:37:05 F.FRANK
 FINAL..1;P777752         17 43155(7)    6-Aug-84 13:45:52 F.FRANK
 OPER.PAS.1;P777752        1 5(7)        5-Aug-84 17:02:10 F.FRANK
   .REL.4;P777752          1 240(36)     6-Aug-84 17:13:05 F.FRANK
 OUTPUT..3;P777752         0 0(36)       2-Aug-84 15:13:02 F.FRANK
 ROOT.FOR.1;P777752        1 107(7)      7-Aug-84 16:01:20 F.FRANK
 SQRT.PGO.2;P777752        1 46(36)      7-Aug-84 09:38:49 F.FRANK
 THIRD.PART.2;P777752      6 13592(7)    4-Aug-84 12:46:50 F.FRANK

 Total of 35 pages in 12 files
@
```

The information provided by the VDIRECTORY is more extensive than that given by the DIRECTORY command. In the output from VDIRECTORY, the notation P777752 signifies the protection number of a file. The file protection number is a file attribute that defines who is allowed which modes of access to the file. The number 777752 permits the file owner total access to the file, and permits other users to read (but not to change) the file. File protections are discussed further, along with other attributes, in section 9.8, page 47.

The next three fields following the file protection are all measures of the file size. The first number is the file size in disk pages; this is the most important measurement of file size, since your disk quota is measured in pages also. A disk page holds 512 computer words or 2560 characters.

The second measure of size is the file byte count and the file byte size. The byte count is the number following the page count; the byte size is the number in parentheses. The byte count is the number of "items" in the file, where the kind of item is specified by the byte size. Common byte sizes are 7, for characters, and 36, for computer words.

More useful than the byte count is the date last written. This is the most recent date and time when the file was written. This tells you when you made the latest change to the file.

The last bit of information provided is the name of the user who last wrote the file. For files on your own directory, this will usually be your own name, but for an unread mail file, for example, it will be the name of the person who last sent you mail.

Both the DIRECTORY and VDIRECTORY commands tell you the total size and number of your files. The number that the directory command prints for your total size is not necessarily the same as the amount of disk space that you really have assigned; see page 27 and page 43.

If you want to see the directory information about a specific file, you may type the file specification of that file after the word DIRECTORY (or VDIRECTORY). Then type RETURN. The DIRECTORY command will show information about that file only.

If you wish to see the directory information about two or more files, type the word DIRECTORY, a space, and the specification of each file, with commas separating adjacent specifications. Type RETURN, not a comma, after the last specification. If a file name alone is used as a specification, the DIRECTORY command interprets that specification to mean all files with the given file name and any file type. An example appears below.

```
@vdirectory bigger,oper.pas

   PS:<F.FRANK>
 BIGGER.FOR.2;P777752      1 82(36)      4-Aug-84 08:10:50 F.FRANK
   .REL.2;P777752          1 1(36)       4-Aug-84 16:56:52 F.FRANK
 OPER.PAS.1;P777752        1 5(7)        5-Aug-84 17:02:10 F.FRANK

 Total of 3 pages in 3 files
@
```

In this example, since the specification "BIGGER" was used, the VDIRECTORY command typed information about all files with the name BIGGER. Because the specification OPER.PAS included a file type, only information about the file OPER.PAS was printed.

9.5. Deleting Files

Getting rid of a file on the DEC-20 takes two steps. First the file must be deleted. Then, deleted files can be *expunged.* Until a file is expunged, the space it occupies counts against your disk quota. The benefit of this two stage process is that you may *undelete* files that were deleted but which have not yet been expunged.

9.5.1. The DELETE Command

The DELETE command in the EXEC is used to delete unwanted files. You would use the DELETE command if your files occupy too much space, or if you were cleaning up files that no longer were useful to you.

To use the DELETE command, type the word DELETE, a space, and a list of file specifications, with commas separating adjacent file specifications. Type RETURN, not comma, at the end of the list of file specifications. Unlike the DIRECTORY command, if a file name alone is used as a specification, no file will be deleted unless a file with the given name and no type exists.

An example should make this clear.

```
@delete sqrt,oper.rel,final
%No such file type - sqrt
 OPER.REL.4 [OK]
 FINAL..1 [OK]
@
```

In this example, although there is a file SQRT.PGO, but it was not deleted because the partial file specification "SQRT" is insufficient in the DELETE command. The other files on the list have been deleted. OPER.REL.4 was deleted because "OPER.REL" is a sufficient file specification to name that file. FINAL..1 was deleted because the lack of file type in the specification "FINAL" matches the empty file type of FINAL..1.

If you were now to type a DIRECTORY command, you would discover that the files FINAL..1 and OPER.REL.4 no longer appear in your file directory.

```
@directory

   PS:<F.FRANK>
 BIGGER.FOR.2
   .REL.2
 DIR..3
 EDIT..2
 OPER.PAS.1
 OUTPUT..3
 ROOT.FOR.1
 SQRT.PGO.2
   .REL.2
 THIRD.PART.2

 Total of 17 pages in 10 files
@
```

9.5.2. Examining your Deleted Files

You can find out which of your files are deleted by using the QDIRECTORY command. QDIRECTORY provides the same sort of information as VDIRECTORY, except that it lists information about your deleted but not yet expunged files.[7]

```
@qdirectory

   PS:<F.FRANK>
 FINAL..1;P777752        17 43155(7)    4-Aug-84 13:45:52 F.FRANK
 MAIL.TXT.1;P775656       0 0(0)        3-Aug-84 14:20:33 F.FRANK
 OPER.REL.4;P777752       1 240(36)     2-Aug-84 15:13:05 F.FRANK

 Total of 18 pages in 3 files
@
```

[7] The MAIL.TXT file, which normally contains messages that people have sent you, is treated specially in the DELETE command. When MAIL.TXT is deleted it is expunged immediately. It is not possible to undelete the MAIL.TXT file. For more information about sending and receiving mail please see page 157.

9.5.3. Undeleting a File

If a deleted file has not yet been *expunged*, it is possible to *undelete* the file. Type the EXEC command UNDELETE, a space, and the specification of the deleted file. It may be necessary to type the generation of the deleted file explicitly in the UNDELETE command.

```
@undelete final..1
 FINAL..1 [OK]
@
```

The UNDELETE command has removed FINAL..1 from the list of deleted files. FINAL..1 now appears in the normal directory.

```
@vdirectory

   PS:<F.FRANK>
 BIGGER.FOR.2;P777752      1 82(36)       4-Aug-84 08:10:50 F.FRANK
   .REL.2;P777752          1 1(36)        2-Aug-84 16:56:52 F.FRANK
 DIR..3;P777752            1 545(7)       4-Aug-84 10:29:10 F.FRANK
 EDIT..2;P777752           4 8273(7)      7-Aug-84 23:37:05 F.FRANK
 FINAL..1;P777752         17 43155(7)     4-Aug-84 13:45:52 F.FRANK
 OPER.PAS.1;P777752        1 5(7)         5-Aug-84 17:02:10 F.FRANK
 OUTPUT..3;P777752         0 0(36)        2-Aug-84 15:13:02 F.FRANK
 ROOT.FOR.1;P777752        1 107(7)       7-Aug-84 16:01:20 F.FRANK
 SQRT.PGO.2;P777752        1 46(36)       7-Aug-84 09:38:49 F.FRANK
   .REL.2;P777752          1 291(36)      7-Aug-84 11:00:07 F.FRANK
 THIRD.PART.2;P777752      6 13592(7)     4-Aug-84 12:46:50 F.FRANK

 Total of 34 pages in 11 files
@
```

Examination of the deleted directory reveals that FINAL..1 has been removed.

```
@qdirectory

   PS:<F.FRANK>
 MAIL.TXT.1;P775656        0 0(0)         3-Aug-84 14:20:33 F.FRANK
 OPER.REL.4;P777752        1 240(36)      2-Aug-84 15:13:05 F.FRANK

 Total of 1 pages in 2 files
@
```

9.5.4. Getting Rid of Deleted Files -- EXPUNGE command

It was mentioned above that the DELETE command merely marks files for later removal. The process of removing the deleted files is called expunging. If you exceed your disk quota during a session, it will be necessary to expunge your deleted files to reclaim disk space. Deleting files does not immediately help reduce your disk space; you must remember to expunge your deleted files.

The EXEC command EXPUNGE may be used to remove all deleted files from your directory. Just type the word EXPUNGE and RETURN to the EXEC's "@" prompt. All files that have already been deleted are permanently removed from your directory; the space that is released is credited to your disk quota.[8]

```
@expunge
 PS:<F.FRANK> [1 page freed]
@
```

Besides the EXPUNGE command, a variety of circumstances may trigger an expunge of your deleted files. Some of these circumstances are:

---

[8] While any program is accessing a file, that file cannot be affected by an EXPUNGE command. If your program stopped with a "disk space full" message, or you typed CTRL/C to interrupt your program, then that program still exists and could be continued by a CONTINUE command. To release the disk space being created by that program, give a RESET command before EXPUNGE.

- When you end a terminal session by the LOGOUT command, the EXEC expunges your directory (and the directory to which you are connected).
- When the computer runs short of disk space, the operating system expunges every user's directory.
- When you exceed your disk allocation, sometimes the program you are running expunges for you.

Because of the variety of circumstances that can trigger an expunge, you should not delete a file with the conviction that you can undelete it. However, if you discover that you have inadvertently deleted a file, you may be able to recover it via the UNDELETE command.

9.6. Making a Copy of a File

Sometimes it is necessary to make a copy of a file. You may have one program that you wish to edit into a new form without losing the original. Your instructor may have a data file that you must copy and change. You may duplicate a file by means of the COPY command.

Type the word COPY, a space, the file specification of the original file, another space, the file specification for the new copy of the file, and press RETURN.

As was discussed above, a file specification may include the directory name of another user. By mentioning the name of another user, you may copy from that person's files (if the file protections permit it).

Warning:

The permissiveness of a user's file protection notwithstanding, it is not proper to copy another person's files without explicit permission.

For example, if you wanted to copy the help file that describes the allocation system, you might type a command such as:

```
@copy hlp:allocations  my.file
 <HELP>ALLOCATIONS.HLP.1 => MY.FILE.1 [OK]
@
```

Most help files reside on the logical device HLP:. Usually, the file name of a help file describes the help topic. This command copies the specified help file to the file named "MY.FILE".

9.7. Changing a File Specification -- RENAME command

You may decide that some file's specification is no longer appropriate, and that you would like to change it. You may change the file specification of a file by the RENAME command.

Type the command RENAME, a space, the old file specification, a space, the new file specification, and RETURN.

One reason why you might need to change the name of a file is that some programs can only handle file names up to six characters (and file types up to three characters). If you had a file named DIAMOND.FOR you would have to think of a six-letter (or shorter) name for it, since "diamond" is too long. Use the RENAME command to fix the file name:

```
@rename diamond.for dia.for
 DIAMOND.FOR.1 => DIA.FOR.1 [OK]
@
```

Due to the way the DEC-20 file system is constructed, it is not possible to use the RENAME command across disk structures. The source and destination file specifications must be on the same structure. You must use the COPY command instead.

9.8. File Attributes

Besides the file specification (the name of the file) and the data contained in the file, there is a variety of other information associated with every file. Some of this is a further description of the file, such as the size of the file in pages, or the date when it was last written. Other information can be explicitly set by the file owner. Some of these other items of information are called *attributes.*

The file attributes that can be set by the file owner include the file's protection, its generation retention count, the account name, the temporary attribute, and the ephemeral attribute.

Although some file attributes appear with the file specification, the attributes are *not* part of the specification. In other words, the file specification "*.*.*;T", where ";T" signifies the temporary attribute, selects ALL files, not just the ones that have the T attribute. Again, this is because the attribute is NOT a part of the file specification.

9.8.1. File Protection Specification

The file protection mechanism is very versatile, allowing the user to define who has access to his files; and what access is allowed. The protection of each file is defined by its file protection number. A file protection number has six digits, divided into three *fields* of two digits each. The protection number is octal, that is, base 8, so only the numbers 0 through 7 are allowed for each digit.

The first two-digit field determines the owner's access. The second field determines the access that group members have. The third field determines the access allowed to all other users.

You may determine the protection number of any file by means of the VDIRECTORY command. The protection number is the six digit number that follows the letter "P" after the file name.

The default file protection varies between the Stanford DEC-20's. The usual protection for files at LOTS is 777752. This protection allows the file owner all modes of access to his files. Other users are permitted to read, but not to modify, the owner's files.

The protection code of 775202 is a common default elsewhere. Again, the owner has all access to the file. Members of a group are allowed to read the file. People who are neither the owner nor a member of the owner's group, however, are unable to read or modify the file; they can only "see" the file in a DIRECTORY command.

The digits in the file protection code have the following meaning:

Digit	Permitting
77	Full access to the file.
40	Read access
20	Write and delete access
10	Execute access
04	Append access
02	The listing of the file specification in a DIRECTORY command.

To permit several modes of access, add the appropriate file protection digits. Thus, for example, to allow reading, writing, and listing of the file in a DIRECTORY command, use code 62.

You can change the file protection by means of the EXEC command SET FILE PROTECTION. Type the command SET FILE PROTECTION, a space, a file specification, another space, the desired file protection number and RETURN. The file specification may include wild-cards, if you wish. For example,

```
@set file protection sqrt.pgo 777700
 SQRT.PGO.2 [OK]
@
```

Warning:

If you set the first two digits of the protection to 00 you will not have access to your own file. To regain access, reset to file protection to 770000 (or whatever) using the generation number 0. If you cannot remember the names of the files you overprotected, contact the staff.

A related command is SET DIRECTORY FILE-PROTECTION-DEFAULT. This command allows you to change the default file protection for all new files that you create. It does not affect existing files, nor does it affect new generations of existing files. This command requires a directory name and protection number as arguments. For example, if Franklin Bell decides that he doesn't want anyone else to access his files, he might give the command:

```
@set directory file-protection-default <f.frank> 777700
@
```

Subsequent to this command, all newly created files will have protection number 777700.

If he also wants to protect the files he has already, he must give the command:

```
@set file protection *.*.* 777700
@
```

Be careful not to over-protect files that you might want others to look at, such as the FINGER.PLAN file.

9.8.2. File Generation Retention Count

Normally, only one generation (version) of each file is kept. Thus, whenever a new version of a file is made, all older versions are deleted. However, the computer system has the facility for keeping more than one generation of any file that you specify. If the count were three, then the three most recent generations would be kept automatically, and the fourth and older generations would be deleted each time a new version is made. This might be useful if you are undertaking a long series of changes, and want to retain older versions of the file for comparison or for safety.

The disadvantage of keeping multiple generations of files is that your disk allocation is used much more quickly.

The EXEC command SET .FILE GENERATION-RETENTION-COUNT may be used to change the number of generations of a file that will be kept. This command takes as its arguments a file specification and a count.

For example:

```
@set file generation-retention-count sqrt.pgo 3
 SQRT.PGO.3 [OK]
@
```

Setting the generation retention count to zero means you want to keep *all* generations of the file. This fills your disk allocation very rapidly.

9.8.3. Account-Name Attribute

The account-name attribute is not currently used at LOTS; on other systems this attribute is used for billing purposes. The account-name file attribute should not be confused with the concept of user name. Unfortunately, in local usage, "user name" is often called "account".

9.8.4. Ephemerals

The ephemeral attribute is applicable to executable files only (file type EXE). When the EXEC runs a program from an EXE file, it looks to see whether the file is marked as an ephemeron. If the file is not an ephemeron, the EXEC will clobber whatever program you may have been running previously. In contrast, if the file is an ephemeron, the EXEC will run it without disturbing your existing program.

The FINGER program, which is run in response to the FINGER command, is an example of an ephemeron. The EDIT

program and the Pascal compiler are examples of non-ephemerons.

An ephemeron is convenient to use because it doesn't disturb other programs that you may be running. Some system utilities are marked as ephemeral. One disadvantage of ephemeral programs is that if you stop one by typing CTRL/C, it cannot be continued.

To mark a file as an ephemeron use the command SET FILE EPHEMERAL, which expects a file specification as an argument. To undo this command, you may use SET FILE NO EPHEMERAL.

9.8.5. Temporary Files

A temporary file specification includes the attribute ";T" and usually a generation number of 100000 plus the job number.

When you log out, the system deletes and expunges all files that have the ";T" attribute and whose generation numbers either match your job number plus 100000 or are smaller than 100000.

To make a file temporary, append ";T" to the file specification when you create or rename it.

9.8.6. REL Files

To execute a program you first create a *source program.* A source program consists of readable characters that people can understand. For example, you may create a file WORK.FOR that contains a FORTRAN source program. The computer cannot deal directly with source programs. A *compiler* is an EXEC program that translates source files into a form that is more suitable for the computer. When you execute a program, the compiler does this translation. Most of the compilers on this system will write a new file containing the translation. The translated file will have the same file name as the source file, but the file type will be REL, meaning *relocatable binary.* Thus, in the case of WORK.FOR, the translation will be stored in WORK.REL.

This translation cannot be run by the computer until it is loaded into main memory. Another program, the loader, performs this process. Once the translation is loaded into memory, the computer can run it. All these steps, translation, loading,and starting, are accomplished by the EXEC command EXECUTE.

It should be noted that the EXECUTE command will avoid translating the program a second time. If there is a REL file that was generated after the source file, then the REL file will be executed. This can create many strange bugs and is occasionally a pain. For example, if you execute the program once, and it is flawed, you will be shown all of your errors. If you try to execute it a second time to see the error messages again, the REL file created from the last execute will take precedence. This REL file will not be executable (because the source file it came from had errors), and you will get the error message NO START ADDRESS. The solution most often is to simply delete the REL file, in which case the source program will take precedence. Another point about REL files is that they tend to take up quite a lot of disk storage. Again, if you need more disk space, you can delete old REL files for which you have the source programs (and they will be created anew if you ever execute the programs again).

Some compilers (for example, PASSGO) combine the translation and loading functions into one process. In such a case, no REL file will be written, and the loading step will be omitted. Such a compiler has two principle advantages: it is generally faster, and it avoids the hassles of the REL file. Of course, there are times when you need REL files, such as when linking your program with a subroutine library.

9.8.7. Command Files

It is often useful to create a file which contains EXEC commands. You can do so by creating a file with the extension .CMD. For example, suppose you often give the command to define your editor to be EMACS. You could create a file with the command that looks like this:

```
define editor: sys:emacs        <-- This says to define my editor as EMACS
set program emacs keep start    <-- This says to keep the fork and start it every time the EXEC command EMACS is given
echo Definitions done!          <-- This prints the message "Definitions done!" on my screen
```

Suppose that these two lines were in a file called FUN.CMD. You could then use the EXEC command TAKE to execute the commands in that file.

```
@take fun
Definitions done!
 End of FUN.CMD.1
```

This has the same effect as directly typing in the commands. If we had include a TAKE command without an argument as the last command in the file, the EXEC would have suppressed the "End of FUN.CMD.1" message.

The above example was rather short; often CMD files are longer. Another advantage of CMD files is that *particular* ones are recognized by the system.

- LOGIN.CMD This file is executed (*taken*) when you log into the system.
- LOGOUT.CMD This file is executed when you log out.
- COMAND.CMD This file is taken whenever you log in or use the PUSH command (see section 13.9).

See Chapter 24, page 161 for a discussion of another type of command file, the MIC file.

10. Terminal Control

The DEC-20 is set up to work with a large variety of terminals. By means of various commands, the EXEC allows you to describe your terminal to the computer.

A display terminal is a delicate beast. Treat it as you would a TV set, with care. Terminals are particularly sensitive to heat and soda pop. Do not place books or papers on top of the terminal, where they impede cooling. Be careful not to spill your soda on the keyboard, either!

10.1. The TERMINAL Commands

The TERMINAL command allows you to set various modes or parameters for your terminal. You may type the EXEC command TERMINAL and any of the following general categories of arguments:

- The specific name of your terminal type. Declaring your terminal type establishes the proper amount of fill and the applicable modes. The list of known terminal types currently includes:

33	Teletype model 33
ADM-3	Lear Seigler ADM-3
AMBASSADOR	Ann Arbor Ambassador
ANN-ARBOR	Old Ann Arbor display
CONCEPT-100	Human Design Concept 100
DATAMEDIA-1520	Datamedia 1520 and 1521 (abbreviated DM1520)
DATAMEDIA-2500	Datamedia 2500 (abbreviated DM2500)
DIABLO	Diablo daisy-wheel letter-quality printing terminal
ESPRIT	Hazeltine Esprit terminal
EXECUPORT	Execuport hardcopy terminal
GILLOTINE	Hazeltine with custom ROM
HAZELTINE-1500	Hazeltine 1500 and 1510
HEATH	Heath (Zenith) H19 or H29 terminal
HP2645A	Hewlett-Packard 264x, 262x, and 2382 series terminals
IBM-3101	IBM 3101 Ascii terminal
LA120	DEC LA120 decwriter-III
LA30	DEC LA30 decwriter
LA36	DEC LA36 decwriter-II
LA38	DEC LA38 decwriter
SOROC-120	SOROC IQ-120
TEKTRONIX-4023	Tektronix 4023 (abbreviated TEK4023)
TEKTRONIX-4025	Tektronix 4024/25/27 (abbreviated TEK4025)
TELERAY-1061	Teleray 1060 series
TELEVIDEO-912	Televideo 912 and 920
TERMINET	GE Terminet printer
TI	TI Silent 700 series
VT05	DEC VT05 terminal
VT50	DEC VT50
VT52	DEC VT52
VT100	DEC VT100
ZENITH	Heath (Zenith) H19 or H29 terminal

- A value of a certain terminal parameter.

LENGTH — Sets the page length of your terminal from the number you type after the word LENGTH.

SPEED — Set the baud rate for terminal input and output. The first number after SPEED is the input baud rate. If the output baud rate is different, type a space after the input baud rate and

then type the output rate. *Use this command* **only** *on dial-up terminals. Using it on a local terminal will cause the terminal to "hang".*

`WIDTH` Set the width of your terminal line from the number following the word `WIDTH`. When a program types a line that is longer than the width of the terminal, the computer will automatically force the output onto the next line of the terminal.

- A specific mode (a mode may optionally be preceded by `NO` to turn it off).

`BACKSPACE-RUBOUT`
Treat the `BACKSPACE` key as a `RUBOUT`. If you clear this mode (`TERMINAL NO BACKSPACE-RUBOUT`) then the `BACKSPACE` key will not erase typing errors; `BACKSPACE` will be seen by programs as a different character than `RUBOUT`.

`BELL` Bells are sent to the terminal to signify end of screen and on some error conditions. To suppress some of these bells, use the `TERMINAL NO BELL` command.

`FLAG` For terminals that have uppercase-only output (`TERMINAL NO LOWER` should be set), print an apostrophe (') before every uppercase character.

`FORMFEED` Terminal has hardware form-feed capability.

`FULLDUPLEX` Treat the terminal keyboard and display separately. This is the normal way that LOTS deals with terminals.

`HALFDUPLEX` Assume that type-in is automatically typed-out by the user's terminal. The alternative to half-duplex is full-duplex. Full-duplex is preferred.

When you are typing on a remote terminal, if you see two characters displayed for each one that you type, your terminal is probably half duplex. Either fix the terminal (there may be a switch) or use this command.

`IMMEDIATE` Echo each character as you type it. Normally, echo is deferred until the program reads the character.

`INDICATE` Print ^L instead of a form-feed.

`LINE-HALFDUPLEX`
Similar to `HALFDUPLEX`, but assumes that the terminal generates a line-feed following each `RETURN` that is typed.

`LOWERCASE` Terminal displays lowercase output properly. If your terminal does not have lowercase display, you may need to give the command `TERMINAL NO LOWERCASE`.

`PAUSE (ON) COMMAND`
Put the terminal in *page mode*. In this mode the terminal will stop typing when you type `CTRL/S`. To resume, type `CTRL/Q`.

`PAUSE (ON) END-OF-PAGE`
Terminal stops when `CTRL/S` is typed, or at the end of each page, where the number of lines in a page is determined by the `TERMINAL LENGTH` command. You may resume type out by typing `CTRL/Q`. For this command to be effective, `TERMINAL PAUSE (ON) COMMAND` must also be set.

`RAISE` Translate lowercase type-in to uppercase. This is useful if you must type uppercase but

have no caps-lock (shift-lock) key. Undo this command by TERMINAL NO RAISE.

TABS Terminal has hardware tab stops every 8 columns.

10.2. Summary of Special Characters

There are a wide variety of control characters. This list summarizes their functions.

CTRL/C Call the EXEC. Interrupt the current program. If the program is waiting for terminal input, one CTRL/C will stop it. Otherwise, two are necessary. Some programs will not allow you to leave by typing CTRL/C. EDIT and BASIC are among these.

CTRL/E Terminate an ADVISE link.

CTRL/F In file-name or command recognition, CTRL/F is similar to ESC, but it completes only one field, instead of completing to the end of the file specification or command.

CTRL/G Bell or beep.

CTRL/H This is the same as BACKSPACE. On the Stanford DEC-20's, BACKSPACE and RUBOUT are normally treated in same manner to delete typing errors. Use of the command TERMINAL NO BACKSPACE-RUBOUT will make BACKSPACE different from RUBOUT. Then, BACKSPACE will be a recognizable character. Some programs need to see backspace. For example, in APL, backspace is used to represent overstrike characters. If you make a typing mistake in some programs (such as the EXEC), you can often recover most of your command line by typing a CTRL/H.

CTRL/I This is the tab character. Tabs are immutably fixed in columns 9, 17, 25, 33, etc. If a terminal has a TAB key, it is equivalent to CTRL/I.

CTRL/J This is the same as the LINE-FEED key. If it matters to you, when you type RETURN the computer automatically adds a line-feed after it. Some programs recognize LINE-FEED as a command.

CTRL/L This is the form-feed character. On the printer, it starts a new piece of paper. CTRL/L also appears between EDIT-style pages in a text file.

CTRL/M This is the same as RETURN. RETURN is often used to signify the end of a line or to *activate* a command.

CTRL/O Typing CTRL/O makes the computer throw away your terminal output. The computer keeps on discarding output until either the program asks for terminal input, or you type a second CTRL/O.

CTRL/Q Allows typeout that was held to resume. Typeout is held by either the screen filling up, or by typing CTRL/S.

CTRL/R Re-display the current input line. Useful if your line got garbled somehow.

CTRL/S Holds terminal output. This only works if the terminal is set to page mode (the default). CTRL/Q will cause output to resume.

CTRL/T Displays job and system status information.

CTRL/U Delete current input line.

CTRL/V Quote the next character. This is useful for getting strange characters into strange places. For example, if you want a file name to contain a question mark, e.g., "WHAT?", you could type "W", "H", "A", "T", CTRL/V, and "?".

CTRL/W Delete the last word on the input line.

CTRL/Z Signals the end of some forms of terminal input.

ESC — In the EXEC and some other programs, the ESC key forces command completion, prompting, and file-name recognition. In other programs, such as EDIT and OPEN, ESC signals the end of multi-line input. ESC is available to all programs for any purpose. In many cases, the program will echo (type) a dollar-sign, "$" in response to ESC.

RUBOUT — This character, which on most terminals is DELETE, usually deletes the character previous to the current cursor position. BACKSPACE also may be used for this function (the BACKSPACE character may be used for other purposes by issuing the TERMINAL NO BACKSPACE-RUBOUT command). RUB, DELETE or DEL are other names for RUBOUT.

11. Care and Feeding of Line Printers

Most line printers on the Stanford DEC-20's are self-service. This means that you are expected to tear off your own listing after it has been printed. There is no operator to load new boxes of paper into the printer. Paper and directions are provided near the printers.

Please treat the printers kindly. They are the only ones we have. When they break, they take a long time to repair. While they are broken, there are no listings.

If the printer jams, the first thing to do is to stop it. Press and release the **On-Line** button at the front of the printer.

It is easy to tear off your own listings. The important thing to remember is *Don't Stop the Paper Movement.* If you hold the paper so that it cannot keep moving out of the printer, the printer will jam. This is a very good way to ruin the listing that follows yours.

Stand directly behind the printer so the paper is moving toward you. At the point where you want to tear it, grasp one edge of the paper with both hands, one hand on either side of the perforation. Be careful not to impede the movement of paper. Pull with the hand that is farthest from the printer to start a tear at the perforation. After starting the tear, shift your grip to one hand on each edge of the paper. Against the resistance of the printer, pull the paper straight back with both hands, pulling harder on the edge where you started the tear. Note that pulling hard on the paper will *not* hurt the printer. Works every time. Perhaps you'd better watch someone else try it.

There is an alternate technique for removing listings. Approach the printer cautiously. Wait until it is busy printing someone else's job. When you see your opportunity, stand directly behind the printer with both feet firmly planted; crouch slightly. Make sure no one is watching. Scream out "Hiiyyaa!" as you grab the flowing paper with both hands and cram it firmly back into the printer, causing a massive paper jam. Rip off your listing, clutch it to your chest and run from the building, yelling "free the DEC 20!" This technique is not highly recommended.

Instructions for solving common line printer problems are posted on the walls near each of the printers. Common tasks which you can perform for yourself include:

- Loading paper. New paper comes in *unopened* boxes located near the printers.
- Adjusting top of form. Occasionally the printer will fail to start printing each new page near the top of the sheet.
- Correcting paper jams. If the printer is printing but the paper is not moving, the paper is probably jammed. You can fix it.
- Replacing the ribbon. If the printer fails to print correctly across the full width of the paper, it may be time to change the ribbon. You should probably ask for assistance before deciding to change the ribbon for yourself.

12. Asynchronous Messages

Occasionally you will find that strange messages appear on your screen. The message may be from another user, or it may be from the computer system. Messages range from mere annoyances to catastrophes.

Although these asynchronous messages seem to mess up your work, usually they are harmless. You may type CTRL/R to re-view your current input line, or you may ignore them completely and continue with your work (unless the message is an ALERT, which you have requested to interrupt your work so you can do something else).

The following examples are fairly common messages.

12.1. Terminal Session Limit Exceeded

On the LOTS computers and the GSB-HOW computer there is a queueing system for ensuring equitable access to the terminals. When there are people waiting in the queue for a terminal, sessions are limited to one and a half hours. Near the end of your session time, you may receive a message like:

```
[Please log off now. (You will be logged off in 5 minutes)]
```

This is a warning that you have only 5 minutes to finish your work. If you ignore this warning, you will receive a message:

```
[Your time is up]
```

whereupon your job will be logged out. If you happen to be in EDIT, any changes that you haven't saved will be wiped out.

12.2. Service Interruptions, Crashes, and Restarts

Sometimes part of the computer system develops a problem. Depending on the nature and severity of the problem, any one of a variety of messages might appear:

```
[DECSYSTEM-20 continued]
```

This message appears after a service interruption. You might not notice any interruption, or you may lose some of the characters that you were typing. Sometimes, when this happens, the listing on the printer will be messed up and may need to be restarted.

```
%DECSYSTEM-20 NOT RUNNING
```

This message, unless followed fairly soon by the continuation message, signifies that the system has crashed. Depending on why the system crashed, it may be up again in 5 minutes, or in extreme cases, not for an entire day. The 5 minute case is by far the more usual.

When the system crashes, you are logged out and lose whatever program is running. If you are editing, you lose whatever changes you made since the last time you saved the file.

```
System restarting, wait...
```

This message signifies that the system is coming up after a crash or other shutdown. Usually it takes two to four minutes to come up after this message is typed.

```
[OPERATOR, TTY126: (to *) Welcome to LOTS.]
```

This message appears after the "restarting" message. When this message appears the system is up and you may log in.

12.3. Messages from Other Users

Every person who uses the DEC-20 can send messages to other terminals. Unfortunately, although this is a useful ability, it may be abused. If you do not want to receive messages from other users, give the command `REFUSE SYSTEM-MESSAGES`.

The `TO` or `SEND` command may be used to send a one line message to another user or to a specific terminal.

Messages that you receive may look like:

```
F.FRANK, TTY27, 7-Aug-84 5:05PM
I propose to head for the Oasis as soon as I can get this Pascal
program working.  Want to come?
```

This message identifies the sender, F.FRANK, and his location, terminal 27, as well as the time it was sent and the text of the message.

You might reply, using the `SEND` command in the EXEC. The destination may be either the user name of the receiver or the terminal number:

```
@send f.frank  Beer! Slurp-slurp!
@send 27 let's do it now!
@
```

See pages 176 and 159 for more details on the SEND command.

12.4. Scheduled Downtime Messages

When the computer system is taken down intentionally for any kind of maintenance activity, there is usually more than 24 hours notice. A forecast of system downtimes is provided by the `INFORMATION DOWNTIME` command.

Sometime before it actually comes down, the system starts warning people with messages like:

```
[System going down in 30 minutes at 30-Aug-84 08:00:00]
```

When the system comes down, it sends a message like:

```
[System down, up again at 30-Aug-84 11:00:00]
```

This message indicates that a planned shutdown has occurred and tells when the system will be up again.

When the system is down, you should look on the screens to see if it says when it will be coming up.

12.5. System Expunge Warnings

When the system runs short of disk space, it will automatically expunge all deleted files from every directory. Before it begins the expunge, it gives a warning:

```
[Caution--Disk space low on PS:]

[Deleted files will be expunged in 5 minutes]
```

At this warning, you should undelete any files you don't want expunged. Then, please delete and expunge any file that you can get rid of.

After four and a half minutes, if people have deleted and expunged enough of their files, the computer may discover that the crisis has passed. It would then say

```
[System expunge postponed]
```

If it is still short of space, the system will announce

```
[Deleted files will be expunged in 30 seconds]
```

Eventually, after the expunge has been completed (it may be several hours if the computer is very busy), the system announces

```
[System expunge completed]
```

13. Summary of EXEC Commands

The following list is provided to briefly explain all of the standard EXEC commands (as of September 1984). The commands are grouped in categories of related use. Although many of these commands have not been described in this manual, they are included here to give you an idea of the full extent and capability of the EXEC.

Note: many of these commands require additional arguments, the form of which may be determined by typing a space then a "?" immediately after the command.

13.1. System Access Commands

These commands allow you to gain and relinquish access to the system, to change your job's account number, and to release and connect terminals to your job. All system access commands except DETACH, ENABLE, and DISABLE may be given without logging in.

ATTACH — Connects your terminal to a designated (already existing) job. If that job was attached to another terminal before you typed the ATTACH command, then the job is first DETACHed, so the other terminal can now be logged in on by any user.

DETACH — Disconnects your terminal from the current job without killing the job. The job will remain idle and will continue to accrue connect charges. The terminal is freed just as if you had logged out.

DISABLE — Returns a privileged user to normal status.

ENABLE — Permits a privileged user to access and change confidential system information. This command changes the EXEC's prompt to a ! instead of the usual @.

INFORMATION ALLOCATION
LOTS only. Displays the weekly limit on computer usage and amount used for a specified user. Subcommands are accessed by following the INFORMATION ALLOCATION with a comma. In subcommand mode the prompt is @@. A useful subcommand is ALL which gives you all the information about your allocation. Subcommands are confirmed by typing a carriage return to the @@ prompt.

INFORMATION DOWNTIME
Displays a schedule of future periods of system unavailability.

INFORMATION QUEUE
LOTS and GSB-HOW only. Displays the queue of people waiting for terminals.

KJOB — Synonym for LOGOUT.

LOGIN — Identifies you and gains access to the DECSYSTEM-20 (See page 6). If a file LOGIN.CMD exists on your directory, the EXEC commands in it are executed as if they had been typed at your terminal.

LOGOUT — Relinquishes access to the DEC-20 and kills your job.

OPEN — LOTS only. Creates a new account, i.e. user name.

SET LOCATION
LOTS only. Sets the "node" or location at which the system thinks your terminal is located. Your location controls such things as where print output will be directed.

UNATTACH — Disconnects a terminal from a job.

13.2. Information Commands

These commands return information about DEC-20 commands, your job, and the system as a whole.

DAYTIME — Prints the current date and time of day.

FINGER — Gives information about logged-in users, including real names, and terminal locations (see page 8).

HELP — Obtains information on almost anything. For a partial list of help topics type HELP ? to the EXEC. Try it. See page 26 for an example.

INFORMATION — Provides information about your job, files, memory, errors, system status, and many other things. For a list of topics, type INFORMATION ?. Information about your job may be obtained with the I JOB command; to see if you have received new mail since you last read it you can use the I MAIL command; I MON displays system performance data, etc.

SET ALERT — Sends a message and a bell to your screen at the time you specify, lest you forget.

SET MAIL-WATCH — Instructs your EXEC to warn you when you receive new mail.

SYSTAT — Outputs a summary of system users and available computing resources.

TRANSLATE — Convert a directory name to or from a project-programmer number for use with TOPS-10 style programs.

13.3. DIRECTORY Command and Variations

These commands list the names of files, and various kinds of information about them.

The DIRECTORY command has a number of useful subcommands. To enter subcommand mode, type an extra comma at the end of the command argument list. The EXEC prompts for subcommands by typing "@@". For a list of subcommands, type ? to the subcommand prompt. Type just RETURN to the "@@" to execute the command. The effects of all commands in this group may be achieved by appropriate choice of subcommands on the DIRECTORY command. For example:

```
@directory,
@@deleted
@@                                   just type RETURN here
```

The following EXEC commands are equivalent to a DIRECTORY command plus various subcommands.

DIRECTORY — Lists in alphabetical order the names of files residing in the specified directory or matching the specified wild-card file specification.

FDIRECTORY — Lists all the information about a file or files. Data includes the name, protection, attributes, number of pages, number of bytes, byte size in parentheses, generations to retain, creation date, date of last write, date of last read, creator, and last writer.

QDIRECTORY — Lists, in VDIRECTORY format, deleted but not yet expunged files.

RDIRECTORY — Lists the names, sizes, and last read date of all files in the order they were last read.

TDIRECTORY — Lists the names of files in the order of the date and time they were last written.

VDIRECTORY — Lists the names of files, as well as their protection, size, date and time they were last written, and the name of the last writer (see page 42).

WDIRECTORY — Lists the names of files, with their write date and author, in the order of the date they were last written.

13.4. File Manipulation Commands

The file system commands allow you to create, delete, and obtain information on files, to specify where they are to be stored, to copy them, and to output them on any device.

`APPEND` Adds the contents of one or more source files to the end of an existing disk file. None of the files should have EDIT line numbers.

`CLOSE` Closes a file or files left open by abnormal termination of a program.

`COPY` Duplicates a source file in a destination file.

`CREATE` Starts your standard editor (normally EDIT) for making a new file.

`DELETE` Marks the specified file(s) for eventual elimination. The file will not appear in your directory, but it will count against your disk allocation until it is expunged (see page 43).

`EDIT` Starts your standard editor (normally EDIT) for changing an existing file (see the chapter on EDIT, page 97). To use a different editor (e.g., a display editor such as EMACS), give the command `DEFINE EDITOR: SYS:EMACS.EXE`. The program named by your definition of `EDITOR:` is the one invoked by your `EDIT` and `CREATE` commands.

`EXPUNGE` Permanently removes any deleted files from the disk.

`INFORMATION FILE-STATUS`
Displays information about your job's currently open files.

`INFORMATION OUTPUT-REQUESTS`
Displays the queue of listings waiting to be printed on the line printer.

`PRINT` Enters a request to list one or more files on the line printer. (see page 38)

`QDIRECTORY` Lists the files which have been deleted, but not yet expunged.

`RENAME` Changes the name of an existing file.

`SET FILE` Sets various characteristics of a file (e.g. protection, ephemeral status, or number of generations to keep).

`SET DEFAULT PRINT`
Allows you to set default switches for your print requests.

`TYPE` Types the specified file(s) on your terminal.

`UNDELETE` Recovers one or more disk files marked as deleted but not yet expunged.

13.5. Directory Access Commands

The commands in this group allow you to obtain information about and change parameters of your directory, to obtain access to other directories, and to change the default directory to be used if you or one of your programs refers to a file without explicitly specifying a directory name.

`ACCESS` Grants ownership and group rights to a specified directory.

`BUILD` Creates a new directory, or changes the parameters of an existing one.

`CD` Synonym for `CONNECT`.

`CONNECT` Changes the default directory name for file specifications, and grants access to it as well as to the logged-in directory (see page 37).

`DEFINE` Creates a logical name which, when used as a device name, will provide default fields for file

specifications.

`END-ACCESS` Relinquishes ownership rights to a specified directory.

`INFORMATION DIRECTORY`
Displays information about a specified directory's privileges and characteristics.

`INFORMATION DISK-USAGE`
Displays the number of pages currently in use and allocated to the connected directory.

`INFORMATION LOGICAL-NAMES`
Displays the definition of the specified logical name. An argument of `SYSTEM` displays the currently defined system-wide logical names, and an argument of `JOB` displays all of your own currently defined logical names.

`INFORMATION STRUCTURE`
Displays information about the specified disk file structure.

`SET DIRECTORY`
Changes the parameters (e.g. password) of a specified directory.

13.6. Device Handling Commands

These commands allow you to reserve a device prior to using it, to manipulate the device, and to release it once it is no longer needed.

`ASSIGN` Reserves a device for exclusive use by your job.

`BACKSPACE` Moves a magnetic tape drive back any number of records or files.

`DEASSIGN` Releases a previously assigned device.

`DISMOUNT` Notifies the system that a mountable device (normally a private disk pack) is no longer needed by your job.

`INFORMATION AVAILABLE DEVICES`
Displays a list of all devices currently known to the system that are available to your job, and all devices which you have assigned.

`INFORMATION AVAILABLE LINES`
Displays a list of the available terminal lines.

`INFORMATION MOUNT-REQUESTS`
Displays a list of all devices which are waiting to be mounted by the operator. Since LOTS normally has no operator on duty, requests in this list will never be serviced.

`INFORMATION TAPE-PARAMETERS`
Displays the job's current default tape format.

`MOUNT` Requests that a mountable device (normally a private disk pack) be made available for your use. If the device is not already present on the system this command waits for operator intervention. Since LOTS normally has no operator on duty, you might find yourself waiting a long time, perhaps forever.

`REWIND` Positions a magnetic tape backward to its load point.

`SET TAPE` Sets default tape format, density, blocksize, and parity.

`SKIP` Advances a magnetic tape one or more records or files.

`UNLOAD` Rewinds a magnetic tape until the tape is wound completely onto the source reel.

13.7. Program Execution Commands

The following commands help you run and manipulate your own programs. See the sections DEBUGGING COMMANDS and FORK STRUCTURE COMMANDS, below, for other useful commands.

`CTRL/C` One or two `CTRL/C` characters will stop the execution of most programs and return you to the EXEC. `CTRL/C` will not stop a program running in the background; you must use the `FREEZE` command instead.

`COMPILE` Translates a source file into an executable, relocatable binary (`REL`) file. The compiler used and other parameters are determined by switches or by the file type. Resets any non-kept forks, and creates a new one.

`CONTINUE` Resumes execution of a program interrupted by a `CTRL/C` or `FREEZE`. You can `CONTINUE` a program in *background* mode which means that both the program and the EXEC will run simultaneously.

`CREF` This command runs the CREF program to produce a cross-reference listing on the line printer. This command resets any non-kept forks, and creates a new one. To produce a cross-reference listing it is necessary to have used the `/CREF` switch in a `COMPILE` or `EXECUTE` command prior to attempting this `CREF` command. Pascal and Passgo users should not give the `CREF` command, since the compiler automatically runs the cross-reference generator program when `/CREF` is specified on the `COMPILE` command.

`CSAVE` Saves a core image of the program currently in memory for later execution. The program is saved in a compressed format. A program that is saved in this way can be run simply by means of the EXEC command `RUN`. This command is not suitable for saving any kind of text file.

`DEBUG` Takes a source program, translates it, loads it with a debugger, and starts the debugger. Resets any non-kept forks, and creates a new one (for FORTRAN see page 145; PASCAL see page 187).

`EXECUTE` Translates, loads, and begins execution of a program. Resets any non-kept forks, and creates a new one (see page 17).

`GET` Loads a saved core image file into memory. Resets any non-kept forks, and creates a new fork.

`LOAD` Translates a program and loads it into memory. Resets any non-kept forks, and creates a new fork.

`MERGE` Loads an executable file into the current fork and without deleting the current contents of memory.

`R` Runs a system program (i.e. an executable file from logical device `SYS:`). `R FOO` is equivalent to `RUN SYS:FOO`. Resets any non-kept forks, and creates a new fork.

`REENTER` Starts the program currently in memory at the program's alternate entry point.

`RUN` Loads an executable program from a file and starts it at the location specified in the program. Resets any non-kept forks, and creates a new fork.

`SAVE` Copies the contents of the current fork into a file in executable format. Such a file can be executed by a `RUN` command. This command is not suitable for saving any kind of text file.

`SET DEFAULT COMPILE`
Allows you to specify the switches to be invoked by a given file type in the `COMPILE`, `LOAD`, and `EXECUTE` commands.

`START` Begins execution of a program at the location specified in the entry vector.

13.8. Program Debugging Commands

The following commands operate on your current fork, and aid in obtaining information about and debugging the programs it contains. These commands are useful for debugging programs at the machine language level. To debug PASCAL programs, see page 187; for FORTRAN, see page 145.

DDT Restarts a debugger, or merges the debugging program DDT with the current program and then starts DDT.

DEPOSIT Places a value in a particular memory location of the current fork.

EXAMINE Displays (in octal) the contents of a particular memory location in the current fork.

INFORMATION ADDRESS-BREAK
Displays the memory location, if any, at which a breakpoint is set.

INFORMATION ERROR-NUMBER
Displays the text for a specified error number.

INFORMATION MEMORY-USAGE
Displays the memory map for the current fork.

INFORMATION PSI-STATUS
Displays the state of the pseudo-interrupt system for the current fork.

MAP Changes the memory map of the current fork.

SET ADDRESS-BREAK
Specifies a memory location which, if referenced, will cause immediate suspension of your program's execution.

SET CONTROL-C-CAPABILITY
Permits your programs to intercept CTRL/C characters. The command SET NO CONTROL-C-CAPABILITY prevents such interception.

SET NO UUO-SIMULATION
Prevents the use of the *compatability package* (PA1050) to simulate TOPS-10 UUOs. Useful if you want to find where a program uses UUOs.

SET PAGE-ACCESS
Changes the accessibility of pages in the current fork.

SET UUO-SIMULATION
Enables the use of the *compatibility package* that permits execution of TOPS-10 programs.

UNMAP Unmap pages from the current fork.

13.9. Fork Structure Commands

At any given time, your EXEC may have several subsidiary programs halted or running. Each subsidiary program is called a fork and has its own (possibly shareable) address space. The following commands modify the set of separate address spaces in use by your job.

ERUN Loads an executable program into a temporary fork and runs it as an ephemeron, i.e. without destroying the current fork. Programs (executable files) may be marked as *ephemeral*, by means of a SET FILE EPHEMERAL command, in which case they are automatically run as ephemerons.

FORK Sets the current fork. Makes EXEC commands work on a particular fork's address space.

FREEZE Stops the execution of a background fork. The fork may be continued by means of the CONTINUE

command.

INFORMATION FORK-STATUS
: Displays the set of forks inferior to the current EXEC.

KEEP
: Marks the current fork as *kept* so that it will not be replaced when another program is run. Programs may also be marked as *autokept*, in which case they are automatically kept whenever they are run.

NAME
: Set the name of the current fork.

POP
: Stops the current copy of the EXEC and returns control to the previous copy of the EXEC. The existing EXEC, and any forks below it, are lost.

PUSH
: Starts a new copy of the EXEC inferior to the EXEC to which you give this command. During the initialization of the new EXEC it will read commands from the file COMAND.CMD if it is present.

RESET
: Deletes the fork specified by the argument. If there is no argument, all stopped, un-kept forks are deleted.

UNKEEP
: Removes the kept status from the current or specified fork, so that it will be replaced the next time another program is run.

13.10. Terminal Commands

The terminal commands allow you to declare the characteristics of your terminal, clear your screen, etc.

BLANK
: Clears the screen on display terminals.

INFORMATION TERMINAL-MODE
: Displays what the DEC-20 is using as your terminal parameters.

TERMINAL
: Declares the hardware type of terminal you have, and lets you instruct the computer on how to treat the display of characters (see page 51).

TERMINAL HELP
: Describes the meanings of each of the parameters you can set with the TERMINAL command.

13.11. Communication Commands

The commands in this group allow you to send messages to other users and to control linking of your terminal to another user's terminal.

ADVISE
: Allows you to type on another screen. Sends whatever you type on your terminal as input to a job connected to another terminal. Advice can be terminated by typing CTRL/E on the terminal that initiated the advise link.

BREAK
: Clears the terminal link established by the TALK command.

MAIL
: Sends a message to another user, to be read at his or her convenience (see page 157).

RECEIVE LINKS
: Allows your terminal to receive links (via TALK commands) from other users.

RECEIVE ADVICE
: Allows your job to receive advice.

RECEIVE SYSTEM-MESSAGES
: Allows your terminal to receive messages (See page 58).

REFUSE LINKS

Denies TALK links to your terminal.

REFUSE ADVICE
Denies advice to your job.

REFUSE SYSTEM-MESSAGES
Denies messages to your terminal (See page 58).

REMARK — Following the REMARK command, all typing up to the next CTRL/Z is treated as a comment. This command is useful if you have established a TALK link and want your typing to appear only on the screen, without being interpreted as commands to your job.

SEND — Synonym for TO (verbose message form). The message you send is also appended to the target user's file SENDS.TXT.

TALK — Links two terminals so that each user can observe what the other user is doing, without affecting the other user's job.

TO — Sends a message to a terminal or logged in user (see page 58).

13.12. Queueing Commands

These commands allow you to examine and modify the queues of requests waiting to print files on the line printer or to run batch jobs. For more information on the batch system, see also page 93.

CANCEL — Eliminate a BATCH or PRINT request.

INFORMATION BATCH-REQUESTS
Examine the queue of jobs waiting to be run by the batch system.

INFORMATION OUTPUT-REQUESTS
Examine the queue of line printer requests.

MODIFY BATCH
Change the parameters of a batch job.

MODIFY PRINT
Change the parameters of a PRINT request.

PRINT — Enter a request to print one or more files on the line printer.

SET DEFAULT PRINT
Set switches to be used in future PRINT commands.

SET DEFAULT SUBMIT
Set switches to be used in future SUBMIT commands.

SUBMIT — Enters a file into the Batch queue. When it is your job's turn to be run, the commands contained in the file are executed.

13.13. Files of Commands

These commands allow you to define new commands or to execute sequences of commands that have been stored in a file.

DECLARE — Defines for the EXEC a PCL (Programmable Command Language) object. This object may be a variable, a new EXEC command, or any of several other possibilities. See HELP PCL for more information.

DO — Accepts program and EXEC commands from a file, echoing them on the screen. Permits macro

parameter substitution. HELP MIC provides details. The file should normally be of type MIC.

ECHO — Reprints the rest of this line on your terminal (useful for documenting CMD files).

INFORMATION PCL-OBJECTS
Displays a list of all objects defined for use by the Programmable Command Language.

KMIC — Terminates MIC processing initiated by a DO command. Not generally useful.

ORIGINAL — Performs an EXEC command in its normal form, ignoring any redefinitions produced by DECLARE commands.

PRESERVE — Saves a PCL environment in a file for later invokation by the DECLARE command.

SET COMMAND-TRACE
Traces command redefinitions produced by a DECLARE command.

TAKE — Accepts EXEC commands from a file, just as if you had typed in the contents of the file on your terminal. The file should normally have the file type CMD.

UNDECLARE — Undefines a PCL object previously created by a DECLARE command.

Part III
An Overview of the Stanford DEC-20's

As of this writing there are nine DECSYSTEM-2060 computers at Stanford run by six different organizations. Although the emphasis on this section is mostly on the LOTS Computer Facility, we do attempt to give some indication of the nature of the other facilities and who to contact regarding accounts and the like.

14. LOTS Computer Facility

The Low Overhead Time-Sharing (LOTS) Computer Facility provides interactive computing for instruction and unsponsored research at Stanford University. Stanford students and faculty may use LOTS without charge for classwork, unsponsored research, and related educational purposes. People who are not associated with the University are not permitted to use LOTS. Further, the facility may *not* be used for any government funded research, nor for any commercial purpose.

Although there are no charges for the use of the LOTS computers, there are limits to how much may be used by each person. These limits are explained in this chapter.

The LOTS facility currently operates three DECSYSTEM-20 computers, called "LOTSA", "LOTSB", and "LOTSC". If you are eligible, you will be given an account on exactly one of the LOTS computers. All instructions in this manual apply to all three machines.

You should obtain the handout *Introduction to LOTS*, available in the CERAS building lobby or at the Terman Engineering Center, room 104. This handout contains the latest information about LOTS.

LOTS conducts introductory classes during the first two weeks of each quarter. You may obtain the class schedule from the CERAS lobby or at Terman 104. These classes introduce students to LOTS and to some computer concepts, but they are not meant to teach programming. The Computer Science Department offers courses in programming.

14.1. Getting Help at LOTS

In addition to the ways of getting help described in Chapter 6, page 25, there are several groups of people that can be of assistance.

14.1.1. Consultants and TA's

If help from the computer isn't sufficient, you can always try the telephone.

LOTS Consultant (CERAS 125):	497-0325
LOTS Consultant (Terman 104):	497-0523
LOTS Office (CERAS 124):	497-3214
For a good time:	767-2676
Dial-a-prayer:	494-0197

During the school year there is often a student consultant in CERAS 125. Also, there is sometimes a consultant at the Terman building. These students will be happy to help you with problems that you encounter while using LOTS.

You should be aware that the consultants are not present to write programs for you. They aren't necessarily familiar with the particular language or subroutines that you are using. The consultants can help with general problems of how to use the system, how to use EDIT, etc.

Many courses have teaching assistants. If you are having trouble with a program related to a specific course, you should approach your TA for assistance. The introductory CS courses have a specific schedule of hours for consulting at LOTS. You can say "`HELP TA`" to the @ to see the current schedule.

You may also seek help from other students who are using LOTS. Once you have had a bit of experience with the computer, you will find that you have picked up a great deal of information that you can share with other people. Please help them. That person next to you who looks lost looks exactly as you did when you first logged in. Though in most cases it is not proper to do another person's assignment, it is certainly appropriate to teach (or learn from) fellow students.

If you have an uncontrollable urge to be helpful, you too can become a consultant at LOTS. Volunteer today!

14.1.2. The LOTS Staff

The LOTS staff is outnumbered over 1000 to 1 by the LOTS users. If you truly cannot get help any other way, they are available for questions on a limited basis. They are easy to find; just look under the large stack of listings in CERAS Room 124. Be aware that the first question *you* are likely to be asked is, "Did you ask the consultant for help?"

14.2. Console Time Allocations

To allocate the computer fairly among the various people who need to use LOTS, there is a weekly limit on console time. Similar limits on CPU (computer) time and pages printed might be implemented in the future.

Allocations are made using the information that you supply when you open your account. If you need to change this, use the program DBEDIT (see page 75). DBEDIT will display what LOTS knows about you and allow you to add or drop classes or otherwise change your reasons for using LOTS.

Rules about allocations:

1. Logging in is not permitted if you have exceeded your weekly limit.

2. Logging in between 2 AM and 10 AM daily is permitted regardless of your actual usage thus far for the week.

3. When you use the computer between 2 AM and 10 AM, the time you use is *not* charged against your weekly allocation.

4. New allocations of time will be made on Sunday mornings before 10 AM. Any unused allocation from a week will NOT carry into the next week.

5. Allocations are normally based on enrollment in courses requiring the use of LOTS. Course enrollment entitles a student to up to 2 hours per week per unit of credit, depending on the computer intensity of the particular course. For example, a 3 unit course, all of whose work requires use of the computer, will allow you 6 console hours each week. The total allocation is the sum of the allocations for each particular purpose.

6. Other reasons for using LOTS may be accepted to increase your allocation. However, for all such reasons (outside of coursework) you must have a faculty sponsor. Obtain the form from the the bookcase in the lobby of the CERAS building or from Terman Room 104.

7. Sponsorship notwithstanding, the primary purpose of LOTS is to support classwork that requires the use of this facility. Therefore, at times of heavy demand from classwork, other uses may be curtailed.

8. During periods of lighter load, you are encouraged to use the computer for unstructured educational purposes. A minimum weekly allocation provides all users with limited access to the computer irrespective of official reasons for using LOTS. During the first half of each quarter, this minimum is 2 hours per week; during the second half of the quarter it is 0 hours per week; between quarters it is 8 hours per week.

9. Only one account per person is permitted. Use of an account by any person other than the account owner is *not* permitted. The account owner is responsible for all use made of the computer by his or her account. The account owner is responsible for the accuracy of the information that he or she furnishes when requesting a LOTS account.

14.3. The Queueing System

When LOTS is crowded (usually the day before an assignment is due), there will be a line of people waiting for terminals. In order to assure first-come first-served access to the LOTS terminals there is a queue (i.e., line of people waiting) for terminals. The queue is controlled by a program you can communicate with at the specially designated queueing terminals.

This year a new queueing system is being implemented. As of this writing its format has not been finalized. Below are the commands that ran the old queueing system. You can expect the new queueing system to behave similarly.

ENTER followed by your user name
: This command adds your name to the queue. Type your user name, not your real name, as the argument. If you want to enter the Terman queue from CERAS or vice versa, add "/T" or "/C" to specify Terman or CERAS respectively.

DELETE followed by a user name
: This command removes the first occurrence of the specified user from the queue. If you decide not to wait for a terminal, please remove your name from the queue before you leave.

HELP
: Provides information on how to use the queueing system, plus documentation on all commands available at the queueing terminal.

OPEN
: This reserves a terminal for a new user running OPEN to get a new account.

As terminals become vacant, users from the queue are assigned to them.

The EXEC command INFORMATION QUEUE displays the status of the terminal queue on any other terminal. The INFORMATION RESERVATION command displays the user name for whom a terminal is reserved, if any.

When there are people waiting in the queue, terminal sessions are limited to one and a half hours, and no one can use a terminal except the person to whom the terminal is assigned. If you receive a warning that you will be logged out, you should log out and enter the queue to be assigned another terminal.

14.4. Using the DBEDIT Program

DBEDIT is a program you can use to change the information LOTS has on file about you and your use of LOTS. This information includes all the data you gave us when you ran OPEN to create your account, and is used to allocate console time limits. It is essential that you keep this information up-to-date if you want to get your fair share of LOTS.

To run DBEDIT give the EXEC command DBEDIT. DBEDIT then will display what LOTS knows about you and allow you to add or drop classes (ADD or REMOVE commands), change an explanation associated with a reason for using LOTS (CHANGE command), or your relation to the university (RELATION command). In the following example, a student explains what he is doing this quarter.

```
@dbedit

Please wait a minute while I look you up.

Name: Franklin P. Bell  Account: F.Frank
Relation: Undergraduate, 0183-305927
Department: Undeclared
Classes and allocation (in square brackets):
    Who knows?
  Total allocation: the default: 2 hours.
PS disk allocation: 50 working, 25 permanent
Last login 20-Sep-83 10:13AM
Entry last changed 28-Jul-83 11:28AM by R.RMK

*?  Please type one of the following:
ABORT        exit without effecting any changes you've requested
ADD          to add yourself to a class
DEPARTMENT   change department or major if you're a student
EXIT         exit and make any changes
GRIPE        tell the LOTS staff about a bug or problem in this program
LIST-CLASSES list all the classes LOTS knows about
NAME         change personal name
RELATION     change student number, whether faculty/staff, etc.
REMOVE       remove yourself from a class
TYPE         type the information on file about you


*add cs111
```

```
*department
What is your department or major?  To see the list of departments we
know about, type a question mark to the wedge prompt.  The list of
departments is rather long; it fills over four terminal screens.  After
one screen has been typed, your terminal will stop and beep.  To see
the next screen, hold down CTRL and type Q. "None" and "Undeclared" are
acceptable answers.
>english

*exit
Bye
@
```

Frank Bell indicated that he has chosen a major and that he is taking CS111 this quarter. By doing so, he obtained his proper time allocation for this term. Note that he could not anticipate; he had to wait until this quarter had begun to indicate his use for the quarter. At the end of each quarter any out-of-date reasons for using LOTS are automatically removed, but you must explicitly add new reasons!

See page 74 for the rules that explain the basis on which console time allocations are made.

14.5. Restoring Files from System Dumps

The LOTS staff makes a full copy of all files stored on disk every Saturday morning. Each weekday morning a backup is done to make copies of all files which have been written since the previous full dump. Early in each quarter (typically about the beginning of the third week), all accounts that have not been used since the end of the previous quarter are eliminated. Before deleting these accounts, all files on the system are saved on an "archive" tape. Full dump tapes are kept for about two months; "archive" dumps are saved indefinitely. These tapes may be used to restore files which have been lost, either by user accident or by account deletion.

To restore your files, you must first find the correct tape, then run DUMPER.

First ask the consultant for the listing binder (possibly labeled "Archive Tapes") for whatever date you are interested in. This binder contains a listing of the names of all files dumped on the specified date, and on which tape they can be found. Users appear in this list in order of their directory names.[9] Find the listing corresponding to the directory you are restoring from. If the listing is not there, ask a consultant for assistance. Once you have the listing of the files that were dumped, look at the top of the listing page to find out the tape number and date.

Put the binder back on the shelf exactly where you found it, with the date showing.

Next, find and mount the tape. Archive tapes are currently located in Room 127, and are filed by date. Ordinary full and incremental dumps are currently located in racks in the computer room. Ask a consultant for assistance in locating where tapes are stored.

```
Find the correct tape and log in.
@assign mta0:
Mount the tape.
```

If you happen to be restoring files from the first tape in a set, note that we place several bootstrapping files at the beginning of the tape. You must skip over these files before starting DUMPER by giving the EXEC command:

```
@skip mta0: 6                                   Only do this if you are using tape number 1
```

Now, run DUMPER. In our example, we use `O.OLDNAME` as your old user name (the one that was purged), and `N.NEWNAME` as your new user name (which may possibly be the same as `O.OLDNAME`).

---

[9] On archive tapes written before October of 1977, users appear in the order of their directory numbers. If you are interested in one of these tapes, ask for assistance.

```
@dumper
DUMPER>tape mta0:
DUMPER>restore <o.oldname>*.*.* <n.newname>*.*.*
```

See Chapter 31, page 229, for more details on how to use the tape drives.

14.6. Pitfalls at LOTS and How to Climb Out

There are some problems with using the DEC-20 that are unique to the LOTS Facility. We will describe the most common problems below. See Chapter 7, page 27 for a list of other problems and their remedies.

14.6.1. Your Account is Frozen

If, when you try to log in at LOTS, the computer responds that your account is frozen, come visit the LOTS Student Coordinator. Bring your Stanford identification. If you can't find the Coordinator, look for a LOTS staff member.

If your account is frozen the first time you try it, probably the computer failed to find your name and student number in the list of registered students. This happens with surprising frequency, so don't be alarmed; the list the computer has on file is usually incomplete.

14.6.2. User Name Does not Match Reservation

This message indicates that the terminal at which you are trying to log in is reserved for someone else. Go to the queueing terminal and insert your name in the queue. If you were assigned this terminal by the queueing program, then you probably took too long to claim it; you have only five minutes from the time your terminal is assigned to you to log in.

14.6.3. Weekly Console Time-Limit Exceeded

Console time allocations are based on the purposes for which you are using LOTS. If you exceed your time limit, you have the following choices:

1. Wait for Sunday. New weekly allocations begin then.

2. Wait until 2 AM. The computer will let you on -- even if you are over allocation -- between 2 and 10 in the morning.

3. You can try having your allocation increased. It is next to impossible to beg additional time from the LOTS staff unless it is between quarters or early in the quarter. You can get a faculty member to sponsor you for more time. This is free of charge; however, be aware that LOTS is principally for coursework. Other uses may be limited when there is a heavy demand from the classes.

4. You might try to get your instructor to assign shorter problems. You might also try training shrimp to whistle.

5. You might try to budget your time at LOTS better. If you come to LOTS with a precise idea of what you are going to accomplish, and how you plan to do it, then your console time allocation ought to be sufficient. On the other hand, if you try to figure out what to do at a terminal, you may soon run out of time.

15. The Other Stanford DEC-20's

15.1. Graduate School of Business - How and Why

The GSB Computer Facility provides interactive computing for instruction and research at the Graduate School of Business at Stanford. Since there are no direct charges for using the facilities, usage is restricted to Stanford GSB students, staff, and faculty, and students enrolled in GSB courses. The GSB facility may *not* be used for any commercial purpose.

The GSB facility currently operates two DECSYSTEM-20 computers, called "How" and "Why". Depending on your reasons for using the computer, you may have accounts on one or both of these systems. If you plan to be using the system for MBA classwork, your account will be on the How computer. Almost all other uses of the GSB timesharing computers are supported on the Why machine.

Contact the GSB facility staff in Room B5 of the Graduate School of Business for further information on the GSB computers and account policies.

15.2. Electrical Engineering - Sierra

The Sierra computer is a DECSYSTEM-2060 operated by the Electrical Engineering Computer Facility for the Stanford Department of Electrical Engineering. Sierra serves the research and administrative needs of the EE Department. Members of the EE faculty, their graduate students and staff, and their research associates in other departments are eligible for Sierra accounts.

The EECF offices are located in room 176 of the McCullough building. Account request forms and additional information pertaining to the use of Sierra may be picked up at the EECF office or the EE main office. Public terminals and laser printers for Sierra can be found in Durand 080.

15.3. Computer Science - Score

The Score computer is a DECSYSTEM-2060 operated by the Computer Science Department Computer Facility. Score serves the research and administrative needs of the Computer Science Department. Members of the CSD faculty, their graduate students and staff, and their research associates in other departments are eligible for Score accounts.

The CSD-CF offices are located in rooms 030 and 040 of Margaret Jacks Hall. Account request forms and additional information pertaining to the use of Score may be picked up at the CSD-CF office.

15.4. Medical Center - SUMEX

SUMEX-AIM is a national research facility for a growing community of projects, both within and external to Stanford. SUMEX provides computing facilities specifically tuned to the needs of artificial intelligence (AI) research. Currently SUMEX supports a user community comprised of approximately 20 active research projects spanning a broad range of application areas. These include clinical diagnostic decision-making, molecular structure interpretation and synthesis, cognitive and psychological modeling, instrument data interpretation, and the generalization of tools to facilitate building new knowledge-based AI.

The SUMEX offices are located in one story redwood buildings adjacent to the Stanford Medical Center in room TB105. The SUMEX computer resources are located within the Stanford Medical Center. The resource computer facility currently consists of a DEC 2060 running the TOPS-20 operating system. The facility also runs a DEC 2020 system for AI-program testing and demonstration and several Lisp workstations for program development, all within an Ethernet local area network. SUMEX is also connected to the TYMNET and ARPANET data communications networks.

15.5. Center for the Study of Language and Information - Turing

The Turing or CSLI DECSYSTEM-20 serves the user community of the Center for the Study of Language and Information (CSLI). CSLI is an interdisciplinary research center involving Xerox, SRI International, Hewlett Packard, Fairchild Labs, and several Stanford departments, including Philosophy, Linguistics, Mathematics, and Computer Science.

The CSLI administrative offices are in Ventura Hall.

Part IV
Program Descriptions

This section is devoted to concise descriptions of various programs and groups of programs that are available on the DECSYSTEM-20's at Stanford.

These programs include computer languages, such as PASCAL, FORTRAN, BASIC and LISP, statistical packages such as SPSS and MINITAB, and utility programs such as DUMPER and TAPEIO.

Besides the programs mentioned here there are many additional utilities and applications. You may look through the HLP: and DOC: file directories for more information.

For very brief descriptions of a variety of the more popular system programs, give the EXEC command "HELP PROGRAMS", or see Appendix B, page 251.

16. Using APL

The APL Language is a very powerful language for writing programs which utilize vector and matrix operations, such as commonly found in quantitative methods in business, operations research and decision science. Its "Workspace" orientation permits many operations on data without the need to write formal programs by using it as a powerful desk calculator. In addition, because APL is an interpreter rather than a compiler, sophisticated interactive debugging, tracing and program modification are possible.

The APL on the DEC-20 is called APLSF, which stands for APL with System Functions.

16.1. Documentation

Extensive documentation on APL can be found in the *APL Reference Manual*, DEC publication DEC-20-LASFA-A-D[10]. The manual is available for perusal at LOTS in the manual rack at CERAS.

Those unfamiliar with the language might also be interested in Leonard Gilman and Rose, *APL, An Interactive Approach*, second edition. This is probably the best tutorial introduction to APL available. It can be purchased at the Stanford Bookstore.

16.2. How To Run APL

The procedures for calling APLSF are more complicated than for many programs because of the need to properly enable the APL character set on some terminals. Terminals also differ in how typing errors are to be corrected. In addition, the BACKSPACE key, which usually causes character deletion when typing, must be redefined as true backspace under APL to permit overstriking of special symbols. Use the EXEC command TERMINAL NO BACKSPACE-DELETE to accomplish this redefinition.

Once you have started an APL session, you can exit with a)MON or)CONTINUE HOLD command. The interrupt key is CTRL/C rather than ATTN or BREAK. You will normally have to type CTRL/C twice to interrupt a function unless APLSF is waiting for you to type in. Be careful not to type too many CTRL/C's in a row; after five or so APL will return you to the EXEC, from which you may continue your APL session by means of the CONTINUE command.

16.2.1. Terminals without the APL character set

Starting an APL session on non-APL terminals consists simply of informing APLSF that you are on a "TTY" (teletype) style terminal without the APL character set:

```
@aplsf
terminal..tty
HI!: NEW USERS PLEASE TYPE:    )LOAD PUB:NEWUSR
APL-20 STANFORD LOTS APLSF 2(435)
TTY44) 13:18:41 TUESDAY 28-AUG-84 F.FRANK       [4,4753]
CLEAR WS
      -
```

APLSF provides a translation scheme so that users with non-APL terminals can enter the special characters and over-struck characters. APL special characters which are also found on ASCII keyboards are used in the same way as on APL keyboards, except for the tilde and underscore as noted below. The following is a list of the translations for other characters.

---

[10] Available from the DEC Technical Documentation Center, Santa Clara, CA 95051, telephone (408) 984-0200.

Single-strike APL characters

∧	And	&
←	Assignment	_
÷	Divide	%
×	Multiply	#
↑	Take	^
*	Power	*
$	Format (DEC extension)	$
|	Residue (ABsolute value)	.AB
α	ALpha	.AL
⍞	Quad (BoX)	.BX
⌈	CEiling (maximum)	.CE
↓	Drop (Down Arrow)	.DA
⊥	DEcode	.DE
∇	DeL	.DL
∩	Down Union (intersection sign)	.DU
⊤	ENcode	.EN
∈	EPsilon	.EP
⌊	FLoor	.FL
≥	Greater than or Equal	.GE
→	GO to (branch)	.GO
⍳	IOta (index generator)	.IO
Δ	Lower Del	.LD
≤	Less than or Equal	.LE
○	Circle (Large O)	.LO
⊃	Left Union	.LU
≠	Not Equal	.NE
¯	NeGation (high minus)	.NG
~	NoT (tilde)	.NT
ω	OMega	.OM
∨	OR	.OR
⍴	RhO	.RO
⊂	Right Union	.RU
∘	Jot (Small O)	.SO
_	UnderScore	.US
∪	Up Union	.UU

Overstruck APL characters

⍝	Comment (lamp)	"
!	Factorial	!
⍒	Grade Down	.GD
⍋	Grade Up	.GU
⌶	I-Beam	.IB
⍟	LoGarithm	.LG
⍲	NaNd	.NN
⍱	NoR	.NR
⍀	Back expansion	.CB
⊖	Rotate (CiRcle)	.CR
⍁	Back scan	.CS
⌹	Divide Quad (matrix inverse)	.DQ
⍞	Input Quad	.IQ
⍞	Output Quad	.OQ
⎕	OUT (escape from function)	.OU
⍎	Protected Del	.PD
⍂	Quad Del	.QD
⍰	Quote Quad	.QQ
⌽	ReVersal	.RV
⍉	TRanspose	.TR
⍎	Execute	.XQ
⍕	ForMat (standard thorn)	.FM
A	Underscored alphabetics	.ZA
Z	...through...	.ZZ
Δ	Underscored lower del	.Z0

By using the above character sequences on non-APL terminals, you should never need to form overstruck characters. You can erase characters by backspacing or with any of the other standard system editing characters (CTRL/U, CTRL/W, CTRL/R, BACKSPACE, DELETE). Note that CTRL/U and CTRL/R do not echo the standard six-space APL prompt; this is a bug, but one you can get used to. You should also note that since "." is used to introduce the special characters, it cannot be used in other contexts where it is followed by a letter; in particular, if you need to specify a file name in APL, use a comma (",") instead of a period to separate the file name from the file type.

16.2.2. Terminals with APL characters

On a terminal supporting the APL character set, you should turn off the normal system conversion of BACKSPACE to character deletion, and should enable the APL character set on your terminal. When APLSF asks you what kind of terminal you are on, you may reply with any of the following:

response	your terminal
key	ASCII APL key pairing (e.g. DM1521, and most terminals with APL)
bit	ASCII APL bit pairing (e.g. HP 2641A)
la36	DEC LA36 with APL character set option
4013	Tektronix 4013
4015	Tektronix 4015
tty	Any terminal not having an APL font

For types "key" and "bit" you will have to engage the APL character set yourself; for "la36", APLSF will try to send the necessary control characters to shift character sets. For example, for a DECwriter LA37 (LA36 with APL character set) terminal, the following commands are necessary to start APLSF:

```
@terminal la36
@terminal no backspace-delete
@aplsf
terminal..la36                                    At this point, APL tries to turn on the APL character set
HI.: NEW USERS PLEASE TYPE:    )LOAD PUB:NEWUSR
APL-20 STANFORD LOTS APLSF 2(435)
TTY44) 14:21:10 TUESDAY 28-AUG-83 F.FRANK       [4,4753]
CLEAR WS
      _
```

To form overstruck characters on an APL terminal, use the BACKSPACE key as is usual in APL. To correct typing errors, backspace to the appropriate point in the line, and then type LINEFEED (CTRL/J). APLSF will ignore the characters to the right, and will advance the cursor to the next line. Do not use ATTN or BREAK.[11]

16.3. Public Libraries

The APL public library space is pointed to by the logical name PUB:. Once you are inside APLSF, you can refer to workspaces on PUB:, or on any of several numbered libraries, called LIB1:, LIB2:, and so forth. The following commands are legal:

```
)LIB PUB:              to see what public workspaces are available
)LOAD PUB:wsid         to load a public workspace
)LIB LIB1:             to see what workspaces are on LIB 1
```

For more information on public libraries, see the APLSF manual. Several public libraries currently exist, among them:

PUB: User-contributed workspaces to perform a variety of functions.

LIB1: APLSF accounting and system maintenance.

LIB2: Statistical programs, including the University of Alberta APL STATPACK collection. See LIB1:CATLOG for details.

LIB4: Miscellaneous utility programs, including functions to return the date and time, to trace other functions conveniently, and to prepare a cross reference listing of an APL function.

LIB9: A tutorial course on APL programming. See LIB9:CATLOG for details.

LIB16: Common statistical procedures used in the behavioral sciences.

The public libraries are only partially tested and documented; use them at your own risk. If you would like to volunteer to help improve them, other APL users at Stanford would be most grateful.

16.4. File I/O

One standard problem in APL systems is communicating with programs written in other languages or reading files written by other programs. APLSF contains several primitives to make this communication easier.

For simple problems the library PUB:LOADMT contains an easy way to load a data matrix into an APL workspace. To read a matrix from file ABC.DAT into A, simply type

```
)COPY PUB:LOADMT LOADMT
A ← LOADMT 'ABC.DAT'
```

---

[11] You can also use the standard system editing characters (CTRL/U, CTRL/W, CTRL/R, and DELETE). The situation gets a bit confusing, though, if your line contains overstruck characters or if you attempt to mix the two editing styles.

In other words, LOADMT is a monadic function whose argument is the name of the file from which the matrix is to be loaded (in quotes). Notes:

- You can create the matrix using EDIT (no line numbers, however) or EMACS, or can read it from magnetic tape or another program.

- Use the "-" sign for negative numbers. LOADMT converts it to unary minus. The data must have no plus signs and consist only of the characters " 0123456789.E-". Conversion will not occur otherwise. Apart from this conversion, LOADMT uses normal APL free form input.

- Each line must contain the same number of elements (since it is a matrix), and each line must have two or more elements (again since it is a matrix).

- If you specified TTY to the "terminal.." prompt then you must specify the file name using a comma instead of a period separating the parts of the file name (i.e. type LOADMT 'ABC,DAT' to prevent APL from interpreting the ".DA" in "ABC.DAT" as "Down Arrow").

For more complex problems, you will probably want to read the "File System" section of the APLSF manual. In that discussion, keep in mind that almost any file written by a non-APL program is an "ASCII sequential file". The function LOADMT described above is illustrative:

```
     ∇LOADMT[⎕]∇
    ∇  Z←LOADMT FILE;CH;NCASE;NVAR;IN
[1]    CH←⎕ASS FILE,' /AS'
[2]    Z←0⍴0
[3]    NCASE←NVAR←0
[4]   ⍝  READ IN INCHARACTER PRIOR TO REPLACING - WITH ¯
[5]   NEWT:IN←⎕[2]CH
[6]    →((⍴⍴IN)=2)/DONE
[7]    NCASE←NCASE+1
[8]   ⍝ LOOK FOR NORMAL -,_REPLACE WITH UNARY
[9]    IN[('-'=IN)/⍳⍴IN]←'¯'
[10]  ⍝ ⎕VI AND ⎕FI CONVERT TO NUMERIC VECTOR AFTER FILTERING OUT NON-NUMERICS
[11]  STICK:Z←Z,IN←(⎕VI IN)/⎕FI IN
[12]  ⍝  SET NVAR THE FIRST TIME (WHEN NCASE = 1)
[13]   NVAR←NVAR+(NCASE=1)×⍴IN
[14]   →(NVAR=⍴IN)/NEWT
[15]   'CHANGE IN NUMBER OF VARIABLES IN A LINE'
[16]   'CASE NUMBER ';NCASE;'    OLD NO OF VARS ';NVAR;'    NEW ';⍴IN
[17]   'EXECUTION OF FUNCTION LOADMT TERMINATED'
[18]   ⎕CLS CH
[19]   →
[20]  DONE:Z←(NCASE,NVAR)⍴Z
[21]   'NUMBER OF CASES = ';NCASE
[22]   'NUMBER OF VARIABLES = ';NVAR
[23]   ⎕CLS CH
    ∇
```

This function opens and assigns a channel number (at [1]) to the file specified as an argument in ASCII sequential mode. It reads successive lines at [5] using quote-quad input with the TTY character set. Thus, the resulting character string (assigned to *IN*) has lower case letters translated to upper and sequences starting with a period translated to the appropriate APL character. Input ends at the first error (presumably end of file) with a branch to *DONE*. The file is finally closed at line [23].

17. Using BASIC

BASIC is a simple, easy-to-learn language that is often used as a first contact with computers. Once they have mastered BASIC, most people move on to a more powerful language such as Pascal or Fortran. The only thing most experienced programmers will use BASIC for is its matrix manipulation facility, and even most of them will forsake BASIC for APL's superior matrix facilities. There are a number of books to be found in the Stanford bookstore that discuss BASIC, but they are likely to teach you a BASIC slightly different from the one in this chapter.

There are at least three versions of BASIC available at Stanford. LOTS BASIC is the version discussed in detail in this chapter. It is a very simple, fast implementation of the BASIC language[12]. BASIC-PLUS-2 is an extended BASIC, supplied by DEC and documented fully in the BASIC-PLUS-2 Language Manual[13]. This version has extensive file and array features. It also supports long variable names, integer variable types and string arrays. Virtual arrays are possible, as well as true function and subroutine calls with argument lists. It also has a desk calculator mode and useful debugging options. BASIC-10, a DEC-10 implementation of BASIC is also available on some systems. It is documented internally and in the DEC-10 BASIC user's manual. Consult the help files on your system to see precisely which BASIC implementation is available.

17.1. BASIC Example

For those already familiar with BASIC and who would like a summary of LOTS BASIC commands, skip the following example and see the "Command Summary", page 90.

Here is an example of how you might start BASIC, enter a program and run it, then leave BASIC. In the example, what the user types in is underlined. We start in the EXEC, which prompts with the `@`. The `BASIC` command starts the BASIC program.

```
@basic
Welcome to LOTS-BASIC
For help, type HELP
To exit, type MONITOR
```

BASIC types a greeting message, spaces down a line and stops. We may now begin typing in our program, including the line numbers. At the end of each line we press the `RETURN` key; BASIC will then be ready to accept another line or a command.

```
10 print "Type in a number"
20 input a
30 if a=0 go to 70
40 let s2=a*a
50 print "The square of",a,"is",s2
60 go to 10
70 stop
```

We have finished typing in the program, so we ask BASIC to run it by typing the `RUN` command. BASIC knows this command is not part of the program because it does not begin with a line number. BASIC types out the name of the program and the date and time. The heading "`NONAME`" stems from the fact that we haven't named the program yet.

```
run
NONAME          17:00          10-Aug-84
? NO END INSTRUCTION
TIME:  0.04 SECS.

READY
```

BASIC tells us that the program needs an `END` statement, then it tells us how long the run took. When BASIC types `READY`, it means that it is waiting for another command or a new program line. We need to add an `END` statement, which

---

[12] A BASIC reference card is available at the LOTS office for a small fee.

[13] The *DECSYSTEM-20 BASIC User's Guide* is available from Digital Equipment Corporation, 2525 Augustine Drive, Santa Clara, CA 95051.

is always the last line in a program. We can give it any number greater than 70. The line numbers aren't required to be incremented by tens.

```
100 end
```

Now our program should run, so we try it again. The "?" tells us that we should type something in. We type in a 15 and BASIC tells us the square of it.

```
run

NONAME          17:01           10-Aug-84

Type in a number
? 15
The square of  15          is           225
Type in a number
? 666
The square of  666         is           443556
Type in a number
? 0
TIME:  0.24 SECS.

READY
```

By the way, you probably understood the program as it was written, without any commentary on our part; this illustrates the advantage of BASIC: it is easy to understand. Now let's add one more line.

```
55 print ''And that's the truth!"
```

It makes perfect sense to insert this line between lines 50 and 60, and this is what BASIC does. Let's see what the program looks like now. The "NH" we append to the "LIST" stands for "no header".

```
listnh

10 PRINT ''Type in a number"
20 INPUT A
30 IF A=0 go to 70
40 LET S2=A*A
50 PRINT "The square of",A,"is",S2
55 PRINT "And that's the truth!"
60 GO TO 10
70 STOP
100 END

READY
```

We rerun the program to see the effect of the new line.

```
runnh

Type in a number
? 4
The square of  4           is           16
And that's the truth!
Type in a number
? 0
TIME:  0.19 SECS.

READY
```

Having put so much effort into our program, we would like to keep it around for later use. In saving it, we decide to give it the name TEST.

```
save test

READY
```

In the future, when we are in BASIC and want to retrieve this program, we would use the command "OLD TEST". We have reached the end of our example, so we return to the EXEC.

```
monitor
0
```

We will find our saved program on our disk area under the name TEST.BAS[14].

17.2. Simple File Manipulation

BASIC has the ability to read and write data from and to disk files.

At the beginning of the program include a FILES statement which declares which files you wish to access.

```
5 FILES  DATA1.DAT , DATA2.TXT , , OUTPUT.BAS
```

These statements will define the disk file DATA1.DAT to be number 1, DATA2.TXT to be file number 2, and OUTPUT.BAS to be file number 4. File number 3 is not defined. The files can be accessed by these numbers. You can access a maximum of nine files in this manner. To read data from the files you use an INPUT statement as follows:

```
10 INPUT #1,A,B,N
20 INPUT #2, B$ , C$
30 INPUT #1, P,Q,K
```

These statements will cause data to be read in the same fashion as if it were in a DATA statement in the program except that no DATA statements are needed in the file.

To write to a file you simply use the PRINT command in the same fashion. Before you write into a file the SCRATCH command must be executed. SCRATCH tells BASIC to erase the present contents of the file and to start writing from the beginning of the file.

```
40 SCRATCH #4
50 PRINT #4,A,B,C,A+B+C,A*B*C
60 PRINT #4
70 PRINT #4, '' THAT'S ALL FOLKS... "
```

Line 40 tells BASIC to prepare file number four for output. Line 50 outputs the shown numbers and the results of the shown arithmetic operations. Line 60 will skip a line in the output file. Line 70 puts a string of characters into the file.

To discover whether you are at the end of a file an IF END test is available. An IF END test does not make any sense if it refers to a file that is being written. Doing this will result in an error.

```
75 IF END #1 THEN 100
```

You can start reading a file over again from the beginning by using the RESTORE command.

```
80 RESTORE #1
90 RESTORE #2
```

As an example of a BASIC file manipulation, say we wish to write a program in BASIC to generate 10 random numbers and put them into a file. We also want to check to make sure they were properly entered and to compute their average.

---

[14] Programs saved by BASIC-PLUS-2 have the extension B20.

```
10 FILES Z.DAT
20 SCRATCH #1
30 FOR I=1 TO 10
40   L=L+1
50   PRINT #1,RND(-1)
60   NEXT I
70 RESTORE #1
80 INPUT #1,L,X
90 N=N+1
100 A=A+X
110 IF END #1 THEN 130
120 GO TO 80
130 PRINT "THE NUMBERS HAVE BEEN STORED AND CHECKED "
140 PRINT"THE MEAN OF THE ";N;" NUMBERS IS ";A/N
150 END

RUNNH

THE NUMBERS HAVE BEEN STORED AND CHECKED
THE MEAN OF THE  10  NUMBERS IS  0.464215

TIME:  0.22 SECS.

READY
```

Note a trick. In line 40 we wrote a file that included sequential line numbers. Since line numbers have been included we can LIST the data file in BASIC. Normally a written file would be accessed from the EXEC.

```
OLD Z.DAT

READY
LISTNH


1            0.217873
2            0.696209
3            0.29751
4            0.963794
5            0.463246
6            0.767746
7            0.829399
8            0.181667
9            0.159454
10           6.52568E-2

READY
```

Of course, it isn't particularly useful to be able to use the OLD command on a data file, since we cannot very well give a RUN command!

17.3. BASIC Command Summary

This is is a summary of commands in version 17D of LOTS BASIC.

`CTRL/C` Interrupts the currently executing program and returns to BASIC command level. To return to the EXEC, give the MON command.

`BYE` Logs you completely off the system.

`CATALOG dev:` Lists onto the user's terminal the names of the user's files which exist on the specified device. For example, `CATALOG BAS:` lists the names of all files on `BAS:` i.e. all public BASIC programs.

`COPY dev:filenm.typ > dev:filenm.typ`
Copies the first file onto the second.

`DELETE line number arguments`
Erases the specified lines from core. For example, "DELETE 11,44-212,13" erases line 11, lines 44

through 212, and line 13. If no arguments are specified, an error message is returned. It is not necessary to use the delete command to erase lines. You can erase a line simply by typing its line number and then depressing the return key.

`LIST line number arguments`
Lists the specified lines of the program currently in core onto the user's terminal. The line number arguments are of the form described under the delete command. If no arguments are specified, the entire program is listed.

`MONITOR` Returns to the EXEC. BASIC may be resumed via `CONTINUE`. This is just like `CTRL/C` for most non-BASIC programs. BASIC intercepts `CTRL/C` aside from returning to BASIC command level.

`NEW dev:filenm.typ`
The program currently in core is erased from core and the specified filename is established as the name of the "program currently in core". N.B., at the end of execution of this command, no lines exist in core.

`OLD dev:filenm.typ`
The program currently in core is erased from core and the specified file is pulled into core to become the new "program currently in core".

`QUEUE filenm.typ`
Does not work! No problem though, instead use: `COPY filenm.typ > LPT:`

`RENAME dev:filenm.typ`
Changes the name of the program currently in core to the specified name.

`REPLACE` See SAVE.

`RESEQUENCE` Changes the line numbers of the program currently in core to 10,20,30,.... (Line numbers within lines (as, GO TO 1000) are changed appropriately).

`RUN` Compiles and executes the program currently in core. The command `RUNNH` is equivalent, but runs the program without printing a header.

`SAVE dev:filenm.typ`
Writes out the program currently in core as a file with the specified name. BASIC will return an error message if save attempts to write over an existing file; to write over a file you must type "REPLACE" instead of "SAVE".

`SCRATCH` Erases from core the program currently in core.

`SYSTEM` Exits from BASIC to monitor level. N.B., the contents of core are lost. You must give the EXEC command `REENTER` rather than `CONTINUE`, `START` or `BASIC` to resume the current BASIC program. "MONITOR" is recommended instead of "SYSTEM".

`UNSAVE dev:filenm.typ`
Deletes the specified file. More than one file can be specified; for example, `UNSAVE DSK:ONE.FOR, SX:TEST.BAS`

In the commands above which accept such arguments, if "`dev:`" is omitted, "`DSK:`" is assumed; if "`.typ`" is omitted, "`.BAS`" is assumed. A null file type is indicated by "`.`" with no "`typ`". The SAVE, REPLACE, and UNSAVE commands allow the "`filenm`" part of the argument to be omitted, in which case "`.typ`" must be omitted also and the name and file type of the program currently in core are assumed. To be consistent with other installations, "`filenm***`" is interpreted as "`BAS:filenm.BAS`".

The keywords of commands (CATALOG, LIST, etc.) may be abbreviated to their first three letters. Only the three letter abbreviation and the full word form are legal; intermediate abbreviations such as CATAL, for example, are not allowed. If an intermediate abbreviation is used, the extra letters will be seen as part of the command argument (because BASIC

does not see blank spaces or tabs at command level.). For example, "CATAL SX:" would be seen as a request to catalog the device "ALSX:". An example of a legal abbreviated command is: CAT DOC:

Whenever BASIC finishes executing a command, it types "READY". It does not answer "READY" after deleting a line by the alternate method described under the delete command or after receiving a line for the program currently in core from the user's terminal.

To insert or replace a line in the program currently in core, simply type the line and then press the RETURN key. BASIC distinguishes between lines (which must be stored or erased) and commands (which must be processed) by the fact that a line always begins with a line number whereas commands all begin with letters.

17.4. Additional BASIC Commands

LOTS BASIC has two new statements: RAISE INPUT and NORAISE INPUT. In both statements, the word "INPUT" can be omitted. These permit user programs to avoid having to test uppercase versus lowercase input. The RAISE statement converts all input to upper case; the NORAISE statement goes back to the (default) condition of not converting to uppercase.

LOTS BASIC also has commands called RAISE TYPEIN and NORAISE TYPEIN. These convert typein of programs (only) to upper case, except for text inside of double quotation marks ("). The default is RAISE TYPEIN. This avoids the problem which many have encountered when the computer does not recognize the variable "a" as being the same as the variable "A". If you want these to be different, use the NORAISE TYPEIN command.

18. Using the Batch System

If you have a procedure that you execute frequently, or a program that is long-running and requires little or no human intervention, then you might submit it as a batch job rather than executing it from your terminal. To submit a batch job, you enter the commands you would normally type on your terminal into a file called a batch control file, then give the SUBMIT command. When your batch job is run, you do not have to be present; the batch system creates a job for you, gives the commands stored in your control file, and records the results in a log file.

For further information, consult the DEC manuals *Getting Started with Batch* and *Batch Reference Manual.*

18.1. Writing a Control File

A control file contains commands and text in almost exactly the same form as the corresponding commands would appear if you typed them during an ordinary terminal session. However, each EXEC command or subcommand should be preceded by an "@", and each line of text to be sent to a program must be preceded by a "*".

If you want a control character entered into the file, type a caret (^) followed by the character. For instance, to enter a CTRL/C, type the two characters "^" and "C". If you want a line sent without the terminating carriage return and linefeed characters, precede the line in the control file with an =.

The control file can have any file name, but should have the file type .CTL. The log file has the same name as the control file and the file type .LOG.

In the following example, a batch control file is created which will compile and execute a FORTRAN program. This FORTRAN program prompts for a number from the terminal, so the number that we want to use is included in the control file. The program writes its output to the file FOR20.DAT, so we also include instructions to print this file.

```
@create test.ctl
Input: TEST.CTL.1
00100   @execute simul.for
00200   *1.5E-7
00300   @print for20.dat
00400   $ <--- Press ESC here, not "$".
*e

[TEST.CTL.1]
@
```

18.2. Submitting a Control File to the Batch System

After writing a control file, you can submit it via the SUBMIT command. The batch system puts your job in the waiting line, which is referred to as the batch input queue (as opposed to the line printer output queue). Each time the batch system is able to run another job, it selects one from the batch queue.

```
@submit test
[Job TEST Queued, Request-ID 9, Limit 0:05:00]
@
```

The general syntax of the SUBMIT command is

```
SUBMIT filename switches
```

Among the more commonly used switches are:

/AFTER: date & time

Submits the job, but runs it only after the specified date and time. You may specify the date and time, or just the time (today is assumed), or "+" followed by the number of minutes to wait before running the job.

/OUTPUT:ALWAYS or ERROR or NEVER

Controls whether the log file should be printed in addition to being written to your directory.

Normally, no log file is printed, but you may instruct the system to print unconditionally, or only on errors.

`/RESTART` Restarts the job when the system restarts after a failure. You should not have a job restarted if it is potentially dangerous to your files.

`/TIME:` time Specifies the maximum amount of central processor time the job may use. The argument time is in the form hh:mm:ss (hours:minutes:seconds). If you do not specify a switch, the system uses a limit of five minutes. If your job exceeds this limit, the system prints the message `?TIME LIMIT EXCEEDED` in the log file then kills the job.

18.3. Checking the Status of a Batch Job

If you want to check the progress of your job, give the `INFORMATION BATCH` command. The system prints a list of all the jobs in the batch queue and their status.

```
@information batch

Batch Queue:
Job Name  Req#  Run Time            User
--------  ----  --------  ------------------------
* TEST       9  00:05:00  J.JQJOHNSON
   Started at 11:07:05
  COMP       3  00:25:00  S.SMOHY
  NITELY     2  00:05:00  OPERATOR                /After: 6-Sep-84 02:00
  WEEKLY     1  00:05:00  OPERATOR                /After: 9-Sep-84 05:00
There are 4 Jobs in the Queue (One in Progress)

@
```

The asterisk at the beginning of the line indicates that the TEST job is running. The jobs COMP, NITELY, and WEEKLY are waiting. NITELY has been requested to run after midnight; COMP has requested 25 minutes of cpu time. Incidentally, 25 minutes is a very large amount of cpu time. You should only run a job this big late at night, by including a `/AFTER` switch when you submit the job. Many systems, in fact, will not schedule batch jobs with time limits of greater than five minutes except during the evening. Give the command `HELP BATCH` for details on CPU restrictions on your DEC-20.

To find out what program the running job is using, give the SYSTAT command.

```
@systat j.jqjohnson
 35*     4 SYSTAT    J.JQJOHNSON
 39    154 LINK      J.JQJOHNSON
@
```

Job number 35 is your current job, and job 39 is the batch job that is linking the SIMUL program. You can keep giving `SYSTAT` commands or `INFORMATION BATCH` commands until the job no longer appears. At that time it is finished and you can examine the output stored in the log file.

18.4. Changing the Status of a Batch Job

You can change the switch settings for jobs awaiting execution by means of the `MODIFY BATCH` command. However, once a job has started execution you can no longer modify it. The following instructs the system to wait until Monday at 5 AM before running the TEST job.

```
@modify batch test /after:sunday+5:00
[1 job modified]
@
```

You can remove a batch job from the queue or stop it in the midst of execution by means of the `CANCEL BATCH` command.

```
@cancel batch test
[1 job canceled]
@
```

If a job awaiting execution is removed from the queue, the system reports that it has been "canceled". If it had started execution, the system also reports that the job was in progress.

18.5. Fancy Control File Commands

The DEC-20 batch facility allows for greater flexibility than we've indicated. You can impose some flow of control using labels and special commands. You can also take special actions if a control file terminates abnormally.

You can put a label (one to six letters long) followed by a "::" at the beginning of a line in the control file. Using these labels you can adjust the order in which statements in the control file are executed. There are three special labels:

- `%ERR::` If an error occurs (the first character of any output is a "?"), the batch system starts executing the commands following this label or a `%FIN::` label if there is one preceding. If the `%ERR::` label is reached normally, the commands following it (up to the next `%FIN::` label) are ignored.

- `%FIN::` This label marks the beginning of a group of commands that must always be executed, regardless of whether or not an error has occurred.

- `%TERR::` This label marks a sequence of commands that are to be executed if the batch job's time limit has been exceeded. An extra 10% of the original limit is granted to execute these commands.

The `BACKTO` and `GOTO` commands take labels as arguments and redirect the flow of control appropriately. `BACKTO` assumes the label is on a preceding line; `GOTO` assumes the label is on a succeeding line.

If you use the `/RESTART` switch when submitting a batch job, then the `CHKPNT` command is useful if you don't want to rerun the entire job. Each time a `CHKPNT` command is given, a new restarting point is remembered by the batch system. When the system comes back up, your batch job will start executing at the point where the last `CHKPNT` command was executed.

The `SILENCE` command will turn off output to the log file. `REVIVE` turns output back on.

The command "`IF (ERROR)`" will execute the following statement if the previous statement caused an error. "`IF (NOERROR)`" performs the converse operation.

The command `ERROR` followed by a character, e.g. "`ERROR ?`" is used to set the error character. Any line beginning with that character will be treated as an error. The default error character is "?".

19. Using EDIT

The first part of this manual, "Getting Started on the DEC-20," showed you how to log in, how to use the terminal, and a little about creating and changing files using the EDIT program. EDIT was mentioned only to show how to create files, and to demonstrate crude techniques for changing them.

From reading the first part of this manual, "Getting Started on the DEC-20", you should already have a solid notion of what files are and how the EDIT program allows you to build and change them. Although what you now know about EDIT enables you to create and change files, there are many more commands available in EDIT that will make your editing sessions more productive. The intention of this chapter is to show you the other EDIT commands and to demonstrate techniques that will make the computer do more of the drudgery.

It is neither necessary nor a good idea to read this rather long chapter in one sitting. It takes time and practice before the concepts explained here will sink in. You might pause between sub-sections to practice each set of new commands.

19.1. Putting Your Program, Data or Text Into a File

If you worked through the examples in "Getting Started on the DEC-20", please bear with us through the next few pages. Although there is some repetition of concepts that were discussed there, we have provided more detail in this exposition. First, though, some important preliminaries.

In "Getting Started on the DEC-20" you were told that typing CTRL/C twice will stop a program. Generally that is true, but EDIT cannot be stopped by CTRL/C. If you type CTRL/C, EDIT will type a message that is something like:

```
Yes? (Type H for help):
```

At this point you really should type "C" to continue in the EDIT program. If you insist, you may return to the EXEC by typing "M". If you do so, you may lose whatever changes you were making. After returning to the EXEC you may continue EDIT (if you haven't clobbered it by running another program in the meantime) by means of the EXEC command CONTINUE.

Typing errors can be corrected by using the DELETE, CTRL/W, and CTRL/U keys. These corrections can be applied to (most) EDIT commands and when you are entering text.

The capitalization of EDIT commands is unimportant, except in special places that are noted. When commands appear in the text of this manual they are usually capitalized. When commands appear in examples, they are usually lowercase.

19.1.1. Building a New File - The EXEC Command CREATE

To create a new file, use the EXEC command CREATE. To the "@" prompt of the EXEC, type the command word CREATE, a space, and the file specification of the new file.

For our example we will create a PASCAL program in a file called LOGS.PGO. The program will be a simple one that calculates the common logarithm of 7.

The choice of the file specification, LOGS.PGO, is partly our arbitrary selection and partly dictated by convention. The file specification is based on the arbitrary selection of the name LOGS, signifying something of the nature of the program, and the conventional selection of the file type PGO, signifying a PASCAL program using the fast (PASSGO) compiler.

To the "@" prompt, type CREATE LOGS.PGO and press RETURN.

```
@create logs.pgo
Input: LOGS.PGO.1
00100
```

In response to the CREATE command, the EXEC starts the EDIT program.[15] EDIT types the word "Input:" to say that it is creating a file for us. Also, it types the file specification of the file being created. This file specification includes the generation number one.

EDIT prompts by typing 00100 and a tab character. EDIT now waits for us to type the text of line 100. The prompt 00100 signifies that the EDIT program is in *insert mode.* While in insert mode, EDIT places text that you type at the terminal into the file.

19.1.2. Entering the Contents of a File

After you give the CREATE command, EDIT waits for you to type the text that you want placed in the file. When EDIT prompts with a line number, type the text of the line and press RETURN. Each time you press RETURN, EDIT will prompt again with a new line number. The line numbers ascend with an increment of 100 between consecutive lines.

Warning:

It is important to end each line with RETURN. ***If you try to type a line that is longer than the width of the screen, the computer will display the end of your line on the next line on the screen. This is NOT the same as pressing*** RETURN.

Although EDIT can handle lines longer than 500 characters, it is a bad idea to make a line that is much longer than the width of the screen. If you make a line longer than 132 characters, you will have difficulty printing it on a line printer. If you make a mistake while typing a 500-character line, imagine the effort required to correct it!

When you are finished typing the text of the file (or if you reach a point where you must give an EDIT command), press the ESC key. It is not necessary to press RETURN after pressing ESC. If the line where you press ESC has no characters on it, EDIT will *not* insert an empty line. If the line has characters, EDIT will keep them on that line.

Now, type in the PASCAL program that appears below. Try to make each line exactly right before pressing RETURN. You can correct typing errors, until you press RETURN, by using DELETE, CTRL/W, or CTRL/U.

```
@create logs.pgo
Input: LOGS.PGO.1
00100   PROGRAM Logs (OUTPUT);
00200   VAR answer: REAL;
00300   BEGIN
00400       answer := LOG (7);
00500       WRITELN ('The common logarithm of 7 is ', answer);
00600   END.
00700
*e

[LOGS.PGO.1]
```

At this point we have come to the end of the program. To signal that we are finished entering text, press the ESC key. EDIT will type a dollar-sign, "$", to mark where it saw the ESC, and prompt with a star, "*", on the next line.

---

[15] If the file you specify already exists, the EDIT program will respond that the CREATE command is for new files only.

```
@create sqrt.pgo
?File already exists (new file required) - sqrt.pgo
@
```

If this occurs, select a different name for your new file.

```
@create logs.pgo
Input:   LOGS.PGO.1
00100    PROGRAM logs (OUTPUT);
00200    VAR answer: REAL;
00300    BEGIN
00400        answer := LOG (7);
00500        WRITELN ('The common logarithm of 7 is ', answer);
00600    END.
00700    $                                    <--- Press the ESC key here. Don't type "$".
*
```

The star that EDIT types is the prompt for *command mode.* In command mode, naturally enough, EDIT awaits commands.

19.1.3. Line Numbers

The line numbers that EDIT types at the left margin help you find and reference the lines of your file. When you are entering text, you do not type these numbers yourself; EDIT will type them for you. EDIT likes to number the lines in multiples of 100. A large increment between lines makes it easy to insert additional lines later on.

A line number consists of five digits and a TAB character. Each line number, with its TAB, uses eight visible columns on the terminal. The first character you can type on a line appears on the terminal in column nine. However, most programs know that line numbers should not be considered as part of the file, so they treat the first character that you type as column one. There are some programs that do not understand about line numbers; SPSS, for example, presently is among these. If you use a program that does not properly ignore the line numbers, there are EDIT commands that remove these numbers.

One further exception that needs to be mentioned is that the BASIC language specifically uses the line numbers as part of the text. If you use EDIT to change a BASIC program, do not renumber the lines and do not remove the line numbers.

Warning:

If you are entering data that will be read by a program that expects numbers in particular columns, do not use TAB ***characters to align columns. Particular places where you should avoid*** TAB ***characters are FORTRAN data files and in the data portion of an*** SPSS ***input file.*** *(*TAB*s within SPSS commands are permitted.)*

Using EDIT, large files can be divided into logical subdivisions called *pages.* (These pages are not related to the unit of disk allocation also called "pages", nor do they necessarily correspond to pieces of printer paper.) Each page in EDIT has its own set of line numbers in ascending sequence. Although we will mostly be talking about files that have only one page, the ability to divide a file at convenient places is very useful. Whenever we speak of a line number, such as 300, we mean line 300 on the *current page.* To speak of line 300 on some specific page, include a slash, "/" and the page number. For example, "300/4" means line 300 on page 4. Page numbers are discussed in greater detail in section 19.5.1, page 124.

19.1.4. Saving Your File and Leaving EDIT

To continue the example, we left EDIT in command mode, awaiting a command.

Since you have finished entering the program, you should tell EDIT to save it for you. You must remember that EDIT will not save the file until you tell it to do so. There are a variety of commands in EDIT for saving the file, depending on what you want to do after you save it.

The first of these commands is the Exit command. To the EDIT command prompt, "*", type "E" and press RETURN.

```
@create logs.pgo
Input:  LOGS.PGO.1
00100   PROGRAM logs (OUTPUT);
00200   VAR answer: REAL;
00300   BEGIN
00400       answer := LOG (7);
00500       WRITELN ('The common logarithm of 7 is ', answer);
00600   END.
00700   $                                          <--- Press the ESC key here. Don't type "$".
*e

[LOGS.PGO.1]
@
```

After you type E, EDIT will write the new file. It types the file specification of the file it writes, and returns to the EXEC program. As usual, the EXEC prompts by typing "@".

Other Ways to Save Your File

Before you go on to run this program, recall that we said there are many ways to save the file using EDIT. Let us mention some of these. All of the commands mentioned below save the changes you have made by writing a new (version of the) file.

E Save your file, leave EDIT, and return to the EXEC.

B Save your file, stay in EDIT. B makes a *backup* copy of your changes to date. After saving the file, EDIT prompts for another command.

G Save your file, leave EDIT, and re-run a program. G means *exit and go.* This command performs the same functions as the E command to save your file. After saving the file, instead of returning to the EXEC, the G command re-does the most recent previous EXECUTE command that you tried during this session. This is useful when developing a program; it gets you out of EDIT and back to your program with a minimum of fuss. The G command repeats your most recent EXECUTE, COMPILE, LOAD, or DEBUG command. Note that this command is not the least bit smart about what EXECUTE function to perform; it simply repeats the last one you did. EDIT will not read your mind.

EU Save the file without line numbers and leave EDIT. EU means *exit unnumbered.* Removing the line numbers is necessary for some programs such as SPSS. Never do it to a BASIC program. Subsequently, when you edit an unnumbered file, EDIT will supply new numbers in increments of 100.

EK Save the file without line numbers or page marks, and leave EDIT.

GU,GK Save the file without line numbers, then perform as G.

BU,BK Save the file without line numbers; remain in EDIT. These commands are less useful than B, even if you plan to discard the line numbers later, because when EDIT re-reads the file after the BU or BK, it will supply new and different line numbers.

The most commonly used of these commands are E, B, G, and EU.

You should use the B command frequently. Sometimes the computer crashes. When this happens, all of the changes that you made since the last B command (or since the last time you entered EDIT) are lost. Perform a B command whenever you have accumulated a number of changes that you are pleased with and which you wouldn't want to repeat again. When you feel daring enough to try a new command, you may wish to use B beforehand to protect yourself from any untoward effects it may have.

Leaving EDIT Without Saving the Changes.

If you discover that you have hopelessly botched an editing session, you may use the EQ command to leave EDIT without writing the changes into a permanent file.

EQ stands for Exit and Quit. This command cancels the EDIT session, without making any changes to your file. If you have used the B command, the EQ command cancels only those changes made since the last B command. EQ is an effective escape if you have somehow damaged your file beyond repair. However, use of the EQ command should be thought of as a desperate measure.

If you blunder in EDIT, you can also try the UNDELETE command (see page 43).

19.1.5. Examining and Executing the Program

Having left the EDIT program, you may use the EXEC command TYPE to examine the file that EDIT made. To the EXEC's "@" prompt, type the command word TYPE, a space, the file specification, LOGS.PGO, and press RETURN.

```
@type logs.pgo
00100   PROGRAM logs (OUTPUT);
00200   VAR answer: REAL;
00300   BEGIN
00400       answer := LOG (7);
00500       WRITELN ('The common logarithm of 7 is ', answer);
00600   END.
@
```

To run this program, type the EXEC command EXECUTE, the file specification, and press RETURN.[16]

```
@execute logs.pgo
Stanford IOTS/Passgo 20 [LOGS  ] -- 1..
Runtime:   0: 0.224
[LOGS   execution]
OUTPUT     :    <--- press the RETURN key to send output to the terminal
The common logarithm of 7 is    8.450980E-01
@
```

Here you see that the program has produced one line of output from the WRITELN statement. The E-01 is scientific notation. It indicates that the number is times ten to the minus one, so the answer above is really 0.8450980.

19.2. Changing a File

The previous section explained how to create a new file. If you followed the examples on your terminal, you created and ran a program that calculates and prints the common logarithm of 7. Now we will discuss how to use EDIT to change an existing file.

19.2.1. Starting the EDIT Program

To use the EDIT program to change an existing text file, start EDIT by means of the EXEC command EDIT. Type the command EDIT to the "@" prompt. Follow the word EDIT with a space and the name of the file you want to change. Finish the command by pressing RETURN.

```
@edit logs.pgo
Edit: LOGS.PGO.1
*
```

---

[16] If your program execution looks radically different from this, you may have made an error when you first entered the file. The sections that follow will explain how to repair such errors.

If you have a problem with this program you may have to stop it by typing CTRL/C twice. You may also have to type CTRL/Q if the terminal fills up with typeout.

Assuming that all goes well, the EDIT program starts and types the file specification of the file it will be changing.[17] EDIT then prompts for a command by typing a star, "*". The "*" prompt signifies that EDIT is in *command mode* awaiting a command. This differs from starting the EDIT program with the CREATE command. When the EDIT program is run in response to a CREATE command, EDIT starts in *insert mode*, where it prompts with line numbers.

In command mode, EDIT expects EDIT commands. We have already seen several EDIT commands, such as the E, G, and B commands mentioned above. We must now explain more of the EDIT commands and how to use them to change a file.

Most, though not all, EDIT commands consist of a single letter that specifies the function. For example, the command letter "I" means *insert*, "P" means *print*, etc. Some commands require that you include a line number, or range of contiguous lines, to specify where the function is to be performed. Commands may be typed in either uppercase or lowercase. A command is not performed until you press the RETURN key.

19.2.2. Viewing Lines - P Command

One of the most useful commands in EDIT is the Print command. The Print Command allows you to "print" selected portions of the file on your terminal. Depending on the line range that you request, a single Print command can display one line, several lines, or the entire file.

Many EDIT commands require that you specify the line or range of contiguous lines that you want the command to act on. For example, in the Print command, the line or line range specifies what line or lines to type. The line or range specification follows the command letter P. EDIT has a variety of ways in which you may specify a line or line range. Although we use the Print command to demonstrate these different ways, keep in mind that these different formats apply to many of the commands that we will discuss later on. Lines and line ranges are essential to finding your way around in EDIT.

To display one specific line, type the command letter P, the line number, and press RETURN. In this example, we type the line 00200.

```
*p200
00200   VAR answer: REAL;
*
```

EDIT allows you to omit the leading zeros in 00200 as shown in this example; 200 is equivalent to 00200.

To print a range of lines, type the command letter P, the number of the first line in the range, a colon, ":", and the number of the last line to be printed. Press RETURN to make EDIT perform the command. For example, to type out lines 200 through 500, use the command "P200:500".

```
*p200:500
00200   VAR answer: REAL;
00300   BEGIN
00400       answer := LOG (7);
00500       WRITELN ('The common logarithm of 7 is ', answer);
*
```

---

[17] If the EDIT program cannot find the file you specified in the command, it will respond with an error message. This message indicates that the file you named does not exist. Perhaps you misspelled the file specification. For example, if we had mistyped LOGS.PGO as LUGS.PGO we would see something like:

```
@edit lugs.pgo

File not found: lugs.pgo   Use the CREATE command to create a new file.

@
```

If this happens to you, check the spelling of the file name. Retype the EDIT command with the correct file specification.

To print a specific number of lines, type P, the line number where you want to begin, an exclamation point "!", the number of lines to print, and press RETURN. The command "P200!3" starts at line 200 and prints a total of three lines.

The caret symbol, "^", may be used instead of a line number to signify the first line on the current page. If you have only one page, this would be the first line in the file. You may use a caret in a range specification anywhere that a line number could appear. Forms such as "P^" and "P^!2" and "P^:400" are all legal, as is demonstrated below.

```
*p^
00100    PROGRAM logs (OUTPUT);
*p^!2
00100    PROGRAM Logs (OUTPUT);
00200    VAR answer: REAL;
*p^:400
00100    PROGRAM logs (OUTPUT);
00200    VAR answer: REAL;
00300    BEGIN
00400        answer := LOG (7);
*
```

The star character, "*", may be used instead of a line number to signify the last line on the current page. If you have only one page, "*" means the last line in the file. As with the caret, the star may be used anywhere that a line number is acceptable. The commands "P*" and "P500:*" are legal forms.

```
*p*
00600    END.
*p500:*
00500        WRITELN ('The common logarithm of 7 is ', answer);
00600    END.

The percent sign character, ''%'' is used to indicate the entire file.
This should only be used for very short files.

*p%
00100    PROGRAM logs (OUTPUT);
00200    VAR answer: REAL;
00300    BEGIN
00400        answer := LOG (7);
00500        WRITELN ('The common logarithm of 7 is ', answer);
00600    END.
*
```

The character period, ".", refers to the *current line*. When EDIT starts, the current line is set to 00000 on page 1. Each time you give a command, EDIT performs some work for you. Generally, the last line affected by the previous EDIT command is the *current line* for the next command. For example, if the previous command was one of the forms of the P command, then the last line that was typed is now the current line.

If you have been following these examples, the current line is now line 600, since the previous command was "P%" and line 600 was the last line that was typed.

As with the caret and star, the period can be used at any point where a line number would be acceptable.

```
*p.
00600    END.
*p300
00300    BEGIN
*p100:.
00100    PROGRAM logs (OUTPUT);
00200    VAR answer: REAL;
00300    BEGIN
*
```

When perusing a file, sometimes it is nice to be able to see small chunks in consecutive order. The P command alone, without arguments, helps you do this. P alone is equivalent to "P.!16". That is, it starts at the current line and prints (up to) 16 lines. Naturally, if there are fewer than 16 lines left before the end of the file, it stops prematurely.

```
*p
00300   BEGIN
00400       answer := LOG (7);
00500       WRITELN ('The common logarithm of 7 is ', answer);
00600   END.
*
```

By repeating the command P with no argument, you can view the file in sixteen line chunks.

When we speak of pages in EDIT, the page number appears with a slash, "/" preceding it. Thus, "300/4" means line 300 on page 4. The form "/3" without a line number refers to the entire contents of page 3.

Thus, the command "P/1" tells EDIT to print page 1.

The command "P/." means print the current page. "P/." is synonymous with "P%" for pages with only one file.

```
*p/.

Page 1
00100   PROGRAM Logs (OUTPUT);
00200   VAR answer: REAL;
00300   BEGIN
00400       answer := LOG (7);
00500       WRITELN ('The common logarithm of 7 is ', answer);
00600   END.
*
```

Another very useful command is LINE-FEED; LINE-FEED is one of the keys on the terminal. As a command to EDIT, LINE-FEED means "print the next line".

Pressing ESC when you're at the * prompt will print the line just above the current one. EDIT marks the place where you pressed ESC by typing a dollar-sign, "$".

In the example below we press ESC three times, then LINE FEED four times. Before we press the first ESC the current line is 600, as set by the previous command.

```
*$00500         WRITELN ('The common logarithm of 7 is ', answer);
*$00400         answer := LOG (7);
*$00300     BEGIN
*↓ <--- type LINE-FEED here, not "↓"
00400       answer := LOG (7);
*↓ <--- type LINE-FEED here, not "↓"
00500       WRITELN ('The common logarithm of 7 is ', answer);
*↓ <--- type LINE-FEED here, not "↓"
00600   END.
*↓ <--- type LINE-FEED here, not "↓"

%No such line(s)
*
```

The error message "No such lines" refers to the fact that we tried to step off the end of the file. If you refer to a range that contains no lines, such as "P200:100" or "P150", EDIT will remark that the command is invalid because the range that you specified is empty.

Specifying a forward or backward offset from a given line is possible. The command "P400+2" means print the second line following line 400. In the present case, that would be line 600. "P550-2" would print line 300 (even though line 550 doesn't exist). Constructs such as "^+2" or ".-1" are legal.[18]

P	Print the current line and the next 15 lines, if possible.
P200	Print line 200.
P200:500	Print lines 200 through 500.
P200!3	Print 3 lines, beginning with line 200.

[18] Caution: "^-5" and "*+10" and related constructs are legal but confusing. EDIT will not cross a page boundary using a + or - construction.

P300-2	Print the second line preceding line 300.
P^	Print the first line on the current page.
P*	Print the last line on the current page.
P.	Print the current line.
P/. or P^:*	Print the entire current page.
P%	Print the whole file.
ESC	Print the previous line.
LINE FEED	Print the next line.

19.2.3. Changing a File - An Example

At last, we are going to start changing the file. The example we have worked with is a file containing a PASCAL program that calculates the common logarithm of 7. Using EDIT, we will transform this simple program into a more versatile one that will compute the common logarithm of any number.

When you are changing a file it is important to have a goal. The goal should be the way you want the file to be when you have finished changing it. Here is the program LOGS.PGO as we want it to look:

```
00100   (* This program calculates common logarithms *)
00200   PROGRAM logs (INPUT, OUTPUT);
00300   VAR answer: REAL;
00400       number: INTEGER;
00500   BEGIN
00600       WRITE ('What number would you like the common log of? ');
00700       READ (number);
00800       answer := LOG (number);
00900       WRITELN ('The common logarithm of ', number, ' is ', answer);
01000   END.
```

Here is the program as it is presently.

```
00100   PROGRAM logs (OUTPUT);
00200   VAR answer: REAL;
00300   BEGIN
00400       answer := LOG (7);
00500       WRITELN ('The common logarithm of 7 is ', answer);
00600   END.
```

Now it is time to transform what we have into what we want.

19.2.4. Deleting Lines - D Command

You can delete a line, a contiguous range of lines, or your entire file using the Delete command. The Delete command requires a line number or line range that describes which lines you want to get rid of. The line or line range is specified in any of the forms we used with the P command. (Except that D alone is an error; it does not delete sixteen lines!)

Looking at the file we have and comparing it to the one we want, we discover that the original line 400 has no further use. Delete line 400 by typing the command letter "D", the line number, and pressing RETURN.

```
*d400
1 lines (00400/1) deleted
*
```

Note that EDIT reports that it has deleted one line and it identifies the line as "00400/1", meaning line 400 on page 1.

To verify that it is gone, print lines 300 through 500.

```
*p300:500
00300   BEGIN
00500       WRITELN ('The common logarithm of 7 is ', answer);
*
```

After a D command, the *current line* is set to the last line that you deleted.

Although we do not want to do it now, a range of lines can be deleted just as readily. For example, to delete all the lines between line 100 and line 300, inclusive, you could use the command "D100:300". But, *don't do it!*

Don't do this next one either. If you try to delete an entire page with a "D%" command, EDIT will make sure you mean it by asking:

```
Massive delete ok? (y or n):
```

Answer "Y" (and press RETURN) if you really want to do it. If you don't want to go through with it, type "N" and press RETURN. If a delete command includes an entire page, or parts of more than one page, EDIT will give you this opportunity to change your mind.

19.2.5. Retrieving Deleted Lines

Once you delete a line using EDIT it is gone. There is no "undelete" command by which a line or group of lines can be retrieved. If the deleted lines were never written to a file, they are lost forever. If you want them back you will have to retype them.

If the lines you deleted were part of the file when you started EDIT, they are still in the original generation of the file. You can use the EDIT Copy command to retrieve lines from the original generation of the file back into this version of the file. The Copy command will be explained in section 19.3.7, page 116. Or, you can use the E command to specify writing your current changes to a different file for the present changes. In this way, when you leave EDIT, you can both save the current changes, and avoid clobbering the original file. This feature of the E command will also be explained later.

Or, you can scrap the current changes by means of an EQ command and start over.

19.2.6. Inserting Lines - I Command

The Insert command is used to add new lines into your file.

To insert a line, type the command letter "I" followed by the number of the line you wish to insert and return. EDIT prints the line number and waits for you to type the new line. Here we insert line 450 and verify the new insertion with a print command.

```
*i450
0450        answer := LOG (number);
*p300:500
00300   BEGIN
0450        answer := LOG (number);
00500       WRITELN ('The common logarithm of 7 is ', answer);
*
```

Lines on each page always appear in ascending sequence. Thus, the line 450 we inserted must come (somewhere) after line 300 and (somewhere) before line 500. We could just as well have chosen 350 or 495 instead of 450.

After an insert command, the *current line* will be set to the last line actually inserted. If you press ESC to the line number prompt without inserting anything, the *current line* is unchanged.

A single insert command cannot interleave multiple new lines with old lines. All lines that are inserted by a single insert command will be contiguous.

Next, insert at line 10:

```
*i10
00010   (* This program calculates common logarithms *)
*
```

Examine the file to see what it looks like after these changes:

```
*p%
00010   (* This program calculates common logarithms *)
00100   PROGRAM Logs (OUTPUT);
00200   VAR answer: REAL;
00300   BEGIN
00450       answer := LOG (number);
00500       WRITELN ('The common logarithm of 7 is ', answer);
00600   END.
*
```

If you compare this with our goal you will see that we still need to insert between lines 200 and 300 the declaration of the variable NUMBER as an integer variable. As noted before, you can select any line number between 200 and 300 for the insert command. If you know that you want it right after line 200, though, you can insert at line 200. Since 200 already exists, EDIT will pick an increment half-way between line 200 and the next line.

```
*i200
00250       number: INTEGER;
*
```

When you insert at the end of a file, EDIT allows you to insert multiple lines with only one Insert command. This mode of operation, called *insert mode*, is identical to the way that we saw EDIT start after a CREATE command. When you have finished typing as many lines as you want, you may press ESC to return to command mode.

Here we specify the last line of the file in the insert command. Give the command "I*" and press RETURN. It doesn't matter what you type on the next lines, but you must press ESC to return to command mode. Then, because we don't really want them, you should delete these lines.

```
*i*
00700   These lines are
00800   a demonstration of the EDIT I command,
00900   and insert mode.  These will be deleted soon.
01000   $                                          <---Press the ESC key, don't type "$" here.
*d700:*
3 Lines (00700/1:00900) deleted
*
```

When you give an insert command, EDIT looks at the line number you specified. If there is no such line in the file, EDIT uses your number as the (first) new line. If the line you specified already exists, EDIT selects a higher line number on which to begin the insertion. EDIT selects a higher number as follows:

EDIT adds the *current increment* to the line number that was specified. If the sum is less than the line number of the line immediately following the specified line, or if there is no line on this page following the specified line, then the sum is used as the line number of the first inserted line. Otherwise, a new line number is selected that is halfway between the specified line and the line that immediately follows it.

EDIT remains in insert mode until you press ESC or until you run out of room in which to insert. Running out of room happens when you press RETURN and the sum of the current line number plus the *current increment* equals or exceeds the line number of the next line in the file, or exceeds 99999.

The *current increment* is usually 100. It can be changed for one insert command by typing a comma and the new increment after the I and the line number, e.g., the command "I400,10" starts inserting at or after line 400, with an increment of 10 between lines. The default current increment can be changed from 100 by means of a command in the form: "/INCREMENT:10''.

The command "/INCREMENT:10" is an example of a *switch*. Switches are used to turn specific program functions on and off or to set new values for internal program parameters. Other examples of switches will be presented later. EDIT will type a list of available switches when you type the command "/?". You can also find out the values of parameters by means of the "=" command. For example, "=INCREMENT" will report the current value of the default increment. You may also type "=?" for a list of the parameters for which information is available.

Some specific illustrations follow. Suppose there are lines numbered 300 and 700 with nothing between, and the *current increment* is 100.

- The command "I550" will start inserting at line 550 and allow the lines 550, and 650 to be entered.
- At this point, the command "I300" would allow lines 400, and 500 to be entered.
- Then the command "I300" would allow only line 350 to be added.
- Finally, the command "I300,10" would allow lines 310, 320, 330, and 340 to be added.

To fit several new lines between existing lines, you may set the insert increment as described above. Another way is to tell EDIT how many lines you want to insert. Do this by typing the command "I", the line number at (or after) which you want the new lines, an exclamation point "!", and the number of lines you want to insert. EDIT chooses a reasonable increment and enters insert mode; you may then type in the new lines. As before, EDIT remains in insert mode until you press ESC or until you collide with an existing line.

In the following example, we wish to insert two lines between lines 300 and 450.

```
*i300!2
00350       WRITE ('What number would you like the common log of? ');
00400       READ (number);
*
```

Observe that EDIT selected an increment of 50 between lines. This increment allows room for exactly two lines, which is what we wanted. Occasionally EDIT will select an increment that allows more lines than we specify. In this case, we can return to command mode by pressing ESC.

Later, when our PASCAL program runs, these lines we have added will ask you for a number. Here's what we have so far:

```
*p%
00010   (* This program calculates common logarithms *)
00100   PROGRAM logs (OUTPUT);
00200   VAR answer: REAL;
00250       number: INTEGER;
00300   BEGIN
00350       WRITE ('What number would you like the common log of? ');
00400       READ (number);
00450       answer := LOG (number);
00500       WRITELN ('The common logarithm of 7 is ', answer);
00600   END.
*
```

Because we have done a lot of editing already, we can use the B command to save our work. Type the command letter "B" and press RETURN.

```
*b

[LOGS.PGO.2]
*
```

Note that the generation number has been updated. This will happen every time you save a new version of the file.

19.2.7. Replacing Lines - R Command

Deleting an old line and putting an entirely new line in its place is called a replacement. The Replace command permits you to delete a range of lines, and then type new lines to replace them. To replace a single line, type the command letter R, the line number, and press RETURN. EDIT will delete the old line and prompt by typing the line number. Type the new contents of this line and then press RETURN. EDIT will (usually) return to command mode, after informing you that the old line has been deleted.

Use the R command to make a new line 100. Type the command "R100" as shown below.

```
*r100
00100   PROGRAM Logs (INPUT, OUTPUT);
1 Lines (00100/1) deleted
*
```

The R command combines the functions of the D command and the I command. In the R command, you can combine the range arguments of the delete command with the insertion increment or line count of the insert command.

The first argument following the R is the range of lines to delete. As soon as you press the RETURN key to perform the Replace command, the lines you specified are deleted. Then, an insertion starts at the first line number that you specified as the delete range. If there is no second argument, the current insertion increment is used for the insertion part of the command. If the second argument is present, it must consist of a comma and either an insertion increment, e.g., ",10" or an insertion count, ",13".

The following are all examples of valid R commands:

```
R3000
R500:700,10
R450!3,18
```

If you replace the last line on a page, EDIT remains in *insert mode* until you press ESC (or until you exceed line 99999). If you replace at a point where adjacent lines differ by more than the *current increment*, then EDIT remains in insert mode until you press ESC or until you collide with the next higher existing line.

Now, change the file once again with the replace command.

```
*r500
00500       WRITELN ('The common logarithm of ', number, ' is ', answer);
1 Lines (00500/1) deleted
*
```

Do a P command to examine the results of your recent editing.

```
*p%
00010   (* This program calculates common logarithms *)
00100   PROGRAM Logs (INPUT, OUTPUT);
00200   VAR answer: REAL;
00250       number: INTEGER;
00300   BEGIN
00350       WRITE ('What number would you like the common log of? ');
00400       READ (number);
00450       answer := LOG (number);
00500       WRITELN ('The common logarithm of ', number, ' is ', answer);
00600   END.
*
```

19.2.8. Changing Line Numbers - N Command

You can see we are very close to the goal we set in section 19.2.3, page 105. All of the text is right, but the line numbers are not neat and regular. The Number command can straighten that out. There is no compelling reason to change the numbers at this point, except to serve as an example.

To renumber your file by hundreds, starting at 100, type "N" and press RETURN.

Warning:
BASIC language programs should never be renumbered using EDIT; use the BASIC command RESEQUENCE.

```
*n
*p%
00100   (* This program calculates common logarithms *)
00200   PROGRAM Logs (INPUT, OUTPUT);
00300   VAR answer: REAL;
00400       number: INTEGER;
00500   BEGIN
00600       WRITE ('What number would you like the common log of? ');
00700       READ (number);
00800       answer := LOG (number);
00900       WRITELN ('The common logarithm of ', number, ' is ', answer);
01000   END.
*
```

When you renumber a file, the lines are left in exactly the same order. In this case, the current line is the last line in the file. In other examples of the N command, the last line in the specified range would be the current line.[19] The N command is useful if you don't have enough space between two lines to insert as many lines as you need to. If you have lines 20 and 25, and wish to insert six lines between them, you cannot do it without first renumbering the file. You cannot fit more than four lines between lines 20 and 25.

To renumber your file in increments of other than 100, type "N" followed by your preferred increment. Here we renumber using an increment of twenty. There is no need for you to perform this example, but you may do so. If you do, when you are done, use an N command with no arguments to renumber by 100s.

```
*n20
*p%
00020   (* This program calculates common logarithms *)
00040   PROGRAM Logs (INPUT, OUTPUT);
00060   VAR answer: REAL;
00080       number: INTEGER;
00100   BEGIN
00120       WRITE ('What number would you like the common log of? ');
00140       READ (number);
00160       answer := LOG (number);
00180       WRITELN ('The common logarithm of ', number, ' is ', answer);
00200   END.
*n                                                    Renumber by 100s
*
```

You can also renumber just a part of a file. The command "N2,60:100,80" will renumber lines 60 through 100 by twos, where the first line is numbered 80, the second 82, etc.

Sometimes, when you are using N (and other) commands, you will cause your line numbers to get out of numerical order, though the lines themselves remain in their proper order. When this happens, EDIT will warn you with this message:

```
%Warning -- Sequence numbers out of order, use N command to correct.
```

You should immediately follow that advice; use the N command to correct any sequencing problems. EDIT will have great difficulty doing anything else before renumbering.

If you use too large an increment, or if you have too many lines on one page, you may get an error message saying %WRAP AROUND. Again, the lines are still in their proper order, but EDIT will have difficulty with subsequent commands unless you renumber again. Select a smaller increment, or a smaller starting line number.

If you have more than 999 lines on a page you will have some difficulty using EDIT. If you have a page that large, you should consider using EDIT to break it into several pages. We will discuss pages further in section 19.5.1, page 124.

[19] If you want to change the order of lines in the files, see the Transfer command (page 116) and the Copy command (page 116).

19.2.9. Leaving EDIT and Re-Running Your Program

Here is the logarithm program as it stands now:

```
*p%
00100   (* This program calculates common logarithms *)
00200   PROGRAM Logs (INPUT, OUTPUT);
00300   VAR answer: REAL;
00400       number: INTEGER;
00500   BEGIN
00600       WRITE ('What number would you like the common log of? ');
00700       READ (number);
00800       answer := LOG (number);
00900       WRITELN ('The common logarithm of ', number, ' is ', answer);
01000   END.
*
```

We have met the goal set earlier (page 105). It is now time to run the program. LOGS.PGO was the last program we executed before going into EDIT. We can take advantage of the G command to save our changes in a new generation of the file, leave EDIT and execute the changed program.[20]

```
*g

[LOGS.PGO.3]

Stanford IOTS/Passgo 20 [LOGS  ] -- 1..
Runtime:   0: 0.264
[LOGS   execution]
INPUT      :    <--- Press the RETURN key to indicate input from the terminal
OUTPUT     :    <--- Press the RETURN key to indicate output to the terminal
[INPUT, end with ↑Z: ]
   <--- and press the RETURN key once more to get the program started
What number would you like the common log of? 693
The common logarithm of          693 is     2.840733E+00
@
```

19.3. Changing the File Without Wasting Your Time

The EDIT commands you have learned so far are sufficient to create and modify any file you will ever make. Editing by removing and retyping entire lines, although effective, is quite crude. The great convenience of EDIT is the ability to make small changes without retyping entire lines. The commands presented in this section are more sophisticated; they can make your editing much faster and easier.

As before, we present a goal. Your mission, should you decide to accept it, is to transform LOGS.PGO above into a program that does error checking. Logarithms only exist for positive numbers. The previous program would reach a fatal error if the user typed in a negative number or zero. The new program will check to make sure the number is positive. It will also do natural logarithms as well as common logarithms. This is your goal:

```
00100   (* This program calculates common logarithms and natural logarithms *)
00200   PROGRAM Logs (INPUT, OUTPUT);
00300   VAR answer: REAL;
00400       number: INTEGER;
00500   BEGIN
00600       WRITE ('What number would you like the logs of? ');
00700       READ (number);
00800       IF number <= 0 THEN
00900       BEGIN
01000           WRITELN ('Bad input.  Number being reset to 1');
01100           number := 1;
01200       END;
01300       answer := LOG (number);
01400       WRITELN ('The common logarithm of ', number, ' is ', answer);
01500       answer := LN (number);
01600       WRITELN ('The natural logarithm of ', number, ' is ', answer);
01700   END.
```

---

[20] If you get the error message No Saved Arguments as a result of a G command, it simply means that you have not previously performed an EXECUTF command during this session. Type the EXEC command EXECUTF and the file specification corresponding to your source program.

Now, re-enter EDIT with the EXEC command EDIT. If you are still logged in from having done the work in the previous section, the EXEC will remember the name of the file you were editing. You need to type only the command EDIT and press RETURN. The EXEC will supply the file specification that it remembers from the previous EDIT command.[21]

```
@edit
Edit: LOGS.PGO.3
*
```

19.3.1. Finding a Specific Piece of Text in the File - F Command

Suppose you are trying to change a relatively large file, containing perhaps several hundred lines. You may not know exactly *where* to make a change, but hopefully you know *what* is there now. The F command helps you discover *where* by searching and finding the *what* that you know is somewhere in the file.

For a specific example, suppose you wish to examine every WRITE statement in the file. The Find command is useful for searching the file for *strings* of characters, such as the sequence of letters "W", "R", "I", etc.

To find the next occurrence of a given *string* in the file, start by typing the command letter "F". Then, without typing any leading spaces, type the precise text of the *string* you are seeking, and press ESC. The ESC, which EDIT *echoes* (i.e., types out) as a "$", is your signal to EDIT that you have reached the end of the string. Press RETURN to execute the command.

Here we find the first occurrence of the word "write" in our file.

```
*fwrite$ <--- Press ESC and RETURN at the end of this line, not "$"
00600       WRITE ('What number would you like the common log of? ');
*
```

Warning:

If EDIT types `''F*''` ***instead of finding a line, you didn't press*** ESC ***to end the search string. Type*** CTRL/U ***now. After EDIT types*** `Aborted...`, ***retype the*** F ***command.***

If you make a typing error in the search string, and you haven't yet pressed ESC, ***you may use*** DELETE ***to correct it. If you discover a mistake in the string after you have pressed*** ESC, ***type*** CTRL/U ***to abort the command. Then retype the command.***

A successful search sets the current line to the (last) line that was found. EDIT finds and prints the first line that contains the desired string, "write". If you want to find the next instance of the same string, type just F and press RETURN. EDIT will repeat the search, starting after the current line, using the same search string that you previously specified. Continuing our example, "F" and RETURN will find the next occurence of WRITE:

```
*f
00900       WRITELN ('The common logarithm of ', number, ' is ', answer);
*
```

Notice that this has found a WRITELN statement. This was found because the string "writeln" *does* contain the string "write". To find just WRITE statements, we would have had to search with the string "`write `" (with a space). This would find all and only the WRITE statements provided that WRITE is always followed by a space in your programs.

If the text you specify cannot be found after the current line EDIT will type `%Search Fails`. The *current line* remains unchanged by an unsuccessful search. In this example, we tell EDIT to find "hello".

```
*fhello$
%Search fails
*p.
00900       WRITELN ('The common logarithm of ', number, ' is ', answer);
*
```

---

[21] If the EXEC responds with the message `No Saved Arguments`, retype the EDIT command but include the file specification, LOGS.PGO, this time. If EDIT starts editing the wrong file, type FQ and press RETURN. Then retype the EDIT command with the correct file specification.

19.3.2. Finding Text Within a Specific Range

EDIT stops at the end of the file if it cannot find the text you have typed; the search does not continue back at the beginning of the file. For instance, if you are at line 900 and give an F command to look for the word "what", EDIT starts looking in line 1000 and proceeds through the last line in the file. If "what" only occurs in line 600, EDIT will not print that line since the search started at line 900.

If the string you seek occurs before the current line, you might give a command such as "PA" to move the current line towards the beginning of the file. Alternatively, after typing the search string and pressing ESC, you may specify the line range in which to perform the search. To search for the first occurrence of the text "number" in lines 500 through 800, you would use the command "Fnumber$500:800".

If you use a line range in a find command, a subsequent command consisting of only the letter F will continue using the same range. To repeat the same string with a different range, you may type F, press ESC, the new range, and press RETURN.

Returning to the problem at hand, the transformation of LOGS.PGO, notice that we need to add some lines after the READ statement. Use the find command to search for the word "READ". Once the READ is located, we will know where to add the new lines.

```
*fread$%
00700       READ (number);
*
```

After finding READ on line 700, the current line is set to 700. Use the command "I.!5" to insert five lines immediately following the line that was just found. EDIT selects an increment of 10 to give you enough room to do it.

```
*i.
00710       IF number <= 0 THEN
00720       BEGIN
00730           WRITELN ('Bad input.  Number being reset to 1');
00740           number := 1;
00750       END;
00760   $ <--- Press ESC and RETURN at the end of this line, not "$"
*
```

19.3.3. Changing a Line Without Retyping it - S Command

The Substitute command provides a way of changing part of a line without retyping the entire line. A substitution has two parts, a *search* and a *replacement*. The search part selects which characters to change. The replacement part specifies what change to make.

To replace one piece of text in a line with another piece, type the command letter "S" followed by the original text, an ESC, the replacement text, an ESC, and the number of line in which the change is to take place.

For instance, the line 600 presently looks like:

```
00600       WRITE ('What number would you like the common log of? ');
```

Our goal for line 600 looks like:

```
00600       WRITE ('What number would you like the logs of? ');
```

We have very little to do. We want to replace the string "common log" with the word "logs". The command "Scommon log$logs$600" (where, as usual, we write "$" to signify the ESC key) will replace the string "common log" on line 600 with "logs".

```
*scommon log$logs$600
00600       WRITE ('What number would you like the logs of? ');
*
```

Warning:

Remember that both the search and replacement strings must be terminated by pressing ESC. ***If EDIT types*** S* ***instead of making a substitution, it is because you didn't press*** ESC ***twice on the command line. Type*** CTRL/U ***and try again. The EDIT prompts*** ''S*'' ***and*** ''F*'' ***actually represent complex extensions of the Substitute and Find commands; see the discussion of the /C128 mode starting on page 128.***

Unless you are careful, you can do a lot of damage with a Substitute command. For example, consider the following line:

```
00100    WRITELN ('A big bad cat was here yesterday.');
```

If we wanted the statement to write out "The big bad . . ." instead, we might try the command "Sathe100". If we do this, though, we get a great mess:

```
*sa$the$100
00100    WRITELN ('the big bthed cthet wthes here yesterdthey.');
*
```

This is hardly what we wanted! You must be careful in describing the *search* part, because the substitute command will replace every instance of the sought-for text with the replacement text.

If this happens to you, sometimes it is easy to undo the damage. In this case we could just change "the" back to "a".

```
*sthe$a$100
00100    WRITELN ('a big bad cat was here yesterday.');
*
```

Notice what happened here. Capitalization was ignored for the search string (i.e., when we searched for "a", we matched both the capital and small a's). But capitalization is adhered to for the substitution. Thus, after these two substitute commands, we are not back to the original line (the first "a" is not capitalized).

This will always be true. The Find and Substitue commands always ignore capitalization when they search. Substitute will substitute with the capitalization that you specify, however.

19.3.4. Substituting on More Than One Line

Sometimes you would like to make the same substitution on several lines throughout your file. The Substitute command that does this is identical to the one you just learned, except that the single line number is replaced with a range of line numbers.

To see the power of this substitute command, change the name "number" to "num" throughout the entire program.

```
*snumber$num$*
00400        num: INTEGER;
00600        WRITE ('What num would you like the logs of? ');
00700        READ (num);
00710        IF num <= 0 THEN
00730            WRITELN ('Bad input.  Num being reset to 1');
00740            num := 1;
00800        answer := LOG (num);
00900        WRITELN ('The common logarithm of ', num, ' is ', answer);
*
```

Imagine doing that using only replace commands! We didn't really want to do this. NUMBER is a much better variable name than NUM, and we've done some substituting that we didn't intend. In this case, we're lucky. We can just substitute back.

```
*snum$number$*
00400        number: INTEGER;
00600        WRITE ('What number would you like the logs of? ');
00700        READ (number);
00710        IF number <= 0 THEN
00730            WRITELN ('Bad input.  Number being reset to 1');
00740            number := 1;
00800        answer := LOG (number);
00900        WRITELN ('The common logarithm of ', number, ' is ', answer);
```

Sometimes students find the need to do a global substitution like this. If they choose a variable name like "FILE", for example, they will find that this is a reserved word in Pascal. They have to change their variable name throughout. This is a useful construct to use in such cases.

It is time to examine the file to see what progress we have made:

```
*p%
00100   (* This program calculates common logarithms *)
00200   PROGRAM Logs (INPUT, OUTPUT);
00300   VAR answer: REAL;
00400       number: INTEGER;
00500   BEGIN
00600       WRITE ('What number would you like the logs of? ');
00700       READ (number);
00710       IF number <= 0 THEN
00720       BEGIN
00730           WRITELN ('Bad input.  Number being reset to 1');
00740           number := 1;
00750       END;
00800       answer := LOG (number);
00900       WRITELN ('The common logarithm of ', number, ' is ', answer);
01000   END.
*
```

In view of how much editing you've done, it's certainly time to do a Backup command.

```
*b

[LOGS.PGO.4]
*
```

If you are about to do a complex Substitute command, it is a good idea to do a B command just beforehand. That way, if something goes wrong you can abandon your mistakes by leaving EDIT using an EQ command. Then when you return to EDIT, the file will be as it was before the imprudent substitution.

19.3.5. Extending a Line - X Command

The Extend command is useful when you want to change something near the end of a line. When you perform an Extend command on a line, EDIT makes the line appear rather as though you had just finished typing the text, without yet having pressed RETURN. Any character that you type will be added to the end of the line, thus extending it.

The character DELETE is effective, as usual, for erasing the character at the end of the line. Repeating DELETE deletes more characters. Using DELETE you may delete back into the original line, or correct errors in the extension of the line. Press RETURN when the line looks right.

The modification required in line 100 seems ideal for the Extend command. The present line looks like:

```
00100   (* This program calculates common logarithms *)
```

We want to change it to be

```
00100   (* This program calculates common logarithms and natural logarithms *)
```

Type the command letter "X", signifying the extend command, the line number, 100, and press RETURN. EDIT types out line 100 and leaves the cursor just after the period.

```
*x100
00100   (* This program calculates common logarithms *)
                                                      ↑
```

In this example the arrow, "↑", has been used to indicate the position of the terminal cursor.

Type DELETE to delete the right parenthesis at the end of the line.

```
*x100
00100   (* This program calculates common logarithms * <--- Type DELETE.
                                                      ↑
```

Use the DELETE key again to erase the *.

```
*x100
00100   (* This program calculates common logarithms  <--- Type DELETE.
                                                     ↑
```

Next, type a space, the text "`and natural logarithms *)`" and press RETURN. After you press RETURN, EDIT reverts to command mode.

```
*x100
00100   (* This program calculates common logarithms and natural logarithms *)
*
```

The X command is often simpler than using a substitution; compare the above with "`S*)$ and natural logarithms *)$100`".[22]

To make changes at the end of each of a number of consecutive lines, use a line range with the X command. EDIT will start you at the end of the first line. When you are done with that line, press RETURN. EDIT will advance to the next line and allow you to make changes there.[23]

19.3.6. Moving Lines Within the File - T Command

The Transfer command moves a line or a group of contiguous lines from one place in the file to another. The original source lines do not remain. To use the transfer command, you need to know the location of the group and where you want it put. Use the transfer command by typing the command letter "T", the destination line number, a comma, the source range, and pressing RETURN.

The transfer command will not clobber any existing lines at the destination. If the line you name as the destination already exists, the lines that move will be inserted after the specified destination line. The transfer command will calculate a suitable increment to use when squeezing the source lines into the destination.

For example, if we wanted to move the range of lines from 500 through 700 to between lines 100 and 200, the command might be "`T150,500:700`". In this command "`150`" indicates the destination line, and "`500:700`" is the range of lines that are being moved. There is a comma between the destination line number and the range.

19.3.7. Copying Lines Within Your File - C command

The Copy command is very similar to the transfer command, except that copy duplicates the sources lines, while transfer moves them.

Here we make a copy of lines 800 and 900 and move the copy to follow line 900. We will further transform these lines into the natural logarithm part of the program.

```
*c900,800:900
INCI=00020
*
```

---

[22] For the sake of truth in advertising, we must note that the extend command is actually a simple variant of the Alter command. An explanation of the alter command appears in section 19.4, page 120. Since the extend command actually invokes the alter command, FSC, CTRL/U, and CTRL/W behave rather differently from the way they do in insert mode. If you are ever confused by what happens, you may return to the EDIT command prompt, without having changed the line, by typing FSC and "Q".

[23] If you want to get back to EDIT command mode without going through the entire range, you may press FSC and type "Q". The line where you pressed FSC and typed "Q" will NOT be changed.

Now, let's see what this looks like.

```
*p%
00100   (* This program calculates common logarithms and natural logarithms *)
00200   PROGRAM Logs (INPUT, OUTPUT);
00300   VAR answer: REAL;
00400       number: INTEGER;
00500   BEGIN
00600       WRITE ('What number would you like the logs of? ');
00700       READ (number);
00710       IF number <= 0 THEN
00720       BEGIN
00730           WRITELN ('Bad input.  Number being reset to 1');
00740           number := 1;
00750       END;
00800       answer := LOG (number);
00900       WRITELN ('The common logarithm of ', number, ' is ', answer);
00920       answer := LOG (number);
00940       WRITELN ('The common logarithm of ', number, ' is ', answer);
01000   END.
*
```

To continue this transformation, we must change the LOG function used in line 920 to the LN function so that we get natural instead of common logarithms.

```
*sLOG$LN$920
00920       answer := LN (number);
*
```

We finish our transformation by changing the word "common" to "natural" in line 940.

```
*scommon$natural$940
00940       WRITELN ('The natural logarithm of ', number, ' is ', answer);
*
```

For the sake of neatness, renumber the entire file; then print it.

```
*n
*p%
00100   (* This program calculates common logarithms and natural logarithms *)
00200   PROGRAM Logs (INPUT, OUTPUT);
00300   VAR answer: REAL;
00400       number: INTEGER;
00500   BEGIN
00600       WRITE ('What number would you like the logs of? ');
00700       READ (number);
00800       IF number <= 0 THEN
00900       BEGIN
01000           WRITELN ('Bad input.  Number being reset to 1');
01100           number := 1;
01200       END;
01300       answer := LOG (number);
01400       WRITELN ('The common logarithm of ', number, ' is ', answer);
01500       answer := LN (number);
01600       WRITELN ('The natural logarithm of ', number, ' is ', answer);
01700   END.
*
```

We have reason to be pleased with what we have. Use a G command to save these changes and to re-run the program.

```
*g

[LOGS.PGO.5]

Stanford LOTS/Passgo 20 [LOGS  ] -- 1..
Runtime:   0: 0.295
[LOGS   execution]
INPUT      : <---  Press the RETURN key to indicate input is from the terminal
OUTPUT     : <---  Press the RETURN key to indicate output is to the terminal
[INPUT, end with ↑Z: ]
    <---  Press the RETURN key again to start things up
What number would you like the logs of? 395
The common logarithm of          395 is      2.596597E+00
The natural logarithm of         395 is      5.978886E+00
@
```

19.3.8. Examining and Copying Parts of Another File

Re-enter EDIT, editing the file LOGS.PGO.

```
@edit
Edit: LOGS.PGO.6
*
```

Besides duplicating lines that are already in the file, the Copy command can be used to copy selected lines from another file. There are two forms of the command that copy from another file. In one, you must know in advance the line numbers of lines you want to copy. In the other form of the command, you peruse the source file before you decide what part to copy.

To demonstrate copying from another file, we will use the file SQRT.PGO that you may have created when you were reading the *Getting Started on the DEC-20* section. We will select some lines from SQRT.PGO and add them at the end of our current program.

First, locate the end of the LOGS.PGO file by means of a P* command.

```
*p*
01700   END.
*
```

We will use the command "C1800=SQRT.PGO/S" to peruse SQRT.PGO and then copy lines from it to the end of our file, LOGS.PGO. In the command, "C" means Copy, as usual. The destination line is 1800. The equal-sign, "=", tells EDIT that the copied text will come from another file. The name of the other file, LOGS.PGO, follows the equal-sign. The "/S" switch means we want to *search* in, or peruse, the other file before deciding what to copy.

```
*c1800=sqrt.pgo/s
C*
```

If you no longer have a copy of the final version of SQRT.PGO from "Getting Started on the DEC-20", then you might try the command "C1800=<F.FRANK>SQRT.PGO/S" to peruse and copy from F.FRANK's version of SQRT.PGO. By the time you try this, the system management may have terminated F.FRANK's account or he may have deleted some of his files. If this happens, and you don't have a version of SQRT.PGO, you will be able to derive only vicarious enjoyment of this example.

Upon your pressing RETURN, EDIT *opens* the file SQRT.PGO for reading and begins prompting with "C*" instead of just "*". The "C*" prompt signals that we are perusing a file. While we are perusing, we may use only the passive commands, such as Print and Find. While perusing, C* mode prohibits pernicious perturbations.

In order to choose the text to copy, you may examine the entire file:

```
C*p%
00100   PROGRAM Roots (OUTPUT);
00200   VAR number, square: INTEGER;
00300       root: REAL;
00400   BEGIN
00500       FOR number := 0 TO 10 DO
00600       BEGIN
00650           square := number * number;
00700           root := SQRT(number);
00800           WRITELN (number, square, root)
00900       END
01000   END.
C*
```

After examining the file, we may select any range of lines to copy from. Let us choose lines 400 through 1000.

To leave "C*" mode and return to our original file, type the command E and press RETURN.

```
C*e
Source lines=
```

EDIT now wants to know which lines we plan to copy.

```
C*e
Source lines=400:1000
*
```

We are now back in the original file; the lines have been copied from SQRT.PGO.[24]

You can check the copying. Use a P% command. The file may be large enough to fill the screen; type CTRL/Q to continue type-out.

```
*p%
00100   (* This program calculates common logarithms and natural logarithms *)
00200   PROGRAM Logs (INPUT, OUTPUT);
00300   VAR answer: REAL;
00400       number: INTEGER;
00500   BEGIN
00600       WRITE ('What number would you like the logs of? ');
00700       READ (number);
00800       IF number <= 0 THEN
00900       BEGIN
01000           WRITELN ('Bad input.  Number being reset to 1');
01100           number := 1;
01200       END;
01300       answer := LOG (number);
01400       WRITELN ('The common logarithm of ', number, ' is ', answer);
01500       answer := LN (number);
01600       WRITELN ('The natural logarithm of ', number, ' is ', answer);
01700   END.
01800   BEGIN
01900       FOR number := 0 TO 10 DO
02000       BEGIN
02100           square := number * number;
02200           root := SQRT(number);
02300           WRITELN (number, square, root)
02400       END
02500   END.
*
```

If we had known beforehand that we wanted lines 400 through 1000 of SQRT.PGO, we could have saved ourselves the effort of visiting the file by using the variant of the Copy command that accepts the source range directly:

```
*c1800=sqrt.pgo,400:1000
*
```

Instead of the "/S" after the file name, we can type a comma and the source range.

There is no reason for us to keep the lines we just added to the file. Delete them with the command "D1800:*".

```
*d1800:*
8 Lines (01800/1:02500) deleted
*
```

19.3.9. Listing Parts of A File on the Line Printer - L Command

It is often convenient to list part or all of your file on the line printer while you are still in EDIT. The List command permits this; type the command letter "L" followed by the range specification of the lines that you want to list.

The command L alone, without an argument, lists the entire file. This command requires confirmation:

```
*l
List entire file on Printer.  OK? [Yes or No]:
```

You may type "yes" and press RETURN if you want the listing, or "no" and press RETURN to return to EDIT command mode without making a listing.

If you don't want the EDIT line numbers on your listing file, you may type ",S" after the range specification. For

[24] If you decide that you do not want any lines from the other file, leave C* mode by typing the FQ command. This will return you to the original file without having copied anything.

example, "L,S" or "L900:*,S".

19.4. Changing the Characters in a Line - A Command

The Alter command allows you to make character by character changes in a line. When you give the Alter command, EDIT enters *alter mode*. In alter mode each character takes on a new meaning as an alter mode command.

We will give you an example by changing around line 600. This is how it currently looks:

```
*p600
00600       WRITE ('What number would you like the logs of? ');
*
```

Suppose you had a grammar teacher who was a stickler about ending sentences in prepositions. We would want to reword the above question as, "Of what number do you want the logs?" We will accomplish this by means of the alter command. Type the command letter "A", the line number 600 and press RETURN. EDIT will type the line number of the line we are changing, and wait for alter mode commands.

```
*a600
00600       WRITE ('What number would you like the logs of? ');
00600
          ↑
```

The arrow, "↑", signifies the location of the cursor.

In alter mode, the cursor is a pointer that you can move left and right in the line until you get to the place where you want to make a change. There are basically two kinds of alter mode commands. One set of commands moves the cursor through the line. Another set of commands makes changes at the place where the cursor is pointing.

Alter mode is somewhat difficult to explain by writing about examples. Alter mode uses the *full duplex* capability of the the terminal to a large extent. Full duplex means that the characters you type do not necessarily appear on the screen. You have seen places, e.g., password checking, where the *echoing* of your typing has been suppressed. In alter mode, commands that you type never appear on the screen. The commands you give effect changes on the screen, but the screen always shows (a portion of) the line you are altering.

Because of the difficulty in showing you commands and their effects, we have adopted a different style of example. In our examples, the arrow, "↑" will indicate successive positions of the cursor. The command characters that we typed will appear under the cursor.

In alter mode, each command is a single character. Some commands require another character as an argument. Commands are never visible, but many commands have visible effects. While you are changing the line, the text that is visible is that portion of the line to the left of the current cursor position.

If you type a character that is not an alter mode command, EDIT will beep in response. The erroneous character will have no other effect.

19.4.1. Alter Mode - Move Right and Move Left

The SPACE bar is the first alter mode command to learn. Typing SPACE moves the cursor one character to the right. Of course, you may type SPACE again and move further.

Try typing SPACE eight times:

```
*a600
00600       WRITE ('What number would you like the logs of? ');
00600       WRIT
          ↑↑↑↑↑↑↑↑
          ————————
```

The cursor should advance past one character each time you type SPACE. As the cursor moves right, the characters that it

passes over become visible.

In the diagram above, the arrows represent successive positions of the cursor. The characters under the arrows represent the character that was typed at that position.

Just as the SPACE character moves the cursor right, the DELETE character moves the cursor left. When the cursor moves left, characters will disappear from the screen. They are *not* being deleted.

Both SPACE and DELETE permit a repeat argument. That is, instead of typing SPACE eight times, you can type the character 8 and SPACE. There are no punctuation characters between the number and the command.

19.4.2. Alter Mode - Print the Line

Generally, the alter command makes the portion of the line that is to the left of the cursor visible, while the right end of the line is invisible. If you are ever confused about what characters are at the right end, you may type the command letter "P".

The alter mode P command causes the portion of the line to the right of the cursor to be printed. Then, EDIT advances the screen to the next line and retypes the line up to the point you were at when you typed P.

Try typing P now.

```
00600       WRITE ('What number would you like the logs of? ');
                ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
                P

00600       WRIT
          ↑↑↑↑↑↑↑↑↑
```

The command letter "L" is similar in purpose to the alter-mode command "P". The L command types the remainder of the text line, advances the terminal to the next line on the screen, and then repositions the cursor at the first character of the text line.

19.4.3. Alter Mode - Find Character

We can make the cursor move in large steps by asking EDIT to *find* a character. To find a character on the line that you are altering, type the command letter "F" and then precisely the one character that you want to find. The command letter "F" may be either in uppercase or in lowercase, but the character following the "F" must exactly match the case of the character that you wish to find.

Note that when you type the command letter "F" absolutely nothing happens. The cursor does not move and the "F" is not visible. Nothing will happen until you type the next character. Then, without your having to press RETURN or anything else, EDIT will advance the cursor up to the character you asked to find.

In this example, type precisely the two letters "fW". The cursor moves to the "W" in "What" (if you fail to capitalize the "W", however, it will skip past this point to the "w" in "would"). To say that the cursor is at the "W" means that the "W" would be the next character passed by a SPACE command.

```
*a600
00600       WRITE ('What number would you like the logs of? ');
00600       WRITE ('
                ↑↑↑↑↑
               fW
```

If the character you type after the "F" can not be found on the line, EDIT advances the cursor to the end of the line.

The "F" command never "finds" the particular character to which the cursor is pointing at the moment you type "F". This permits repetitions of the F command to find successive instances of a character.

19.4.4. Alter Mode - Insert New Characters

At this point, we have to add the word "Of" and a space. Type the alter mode command I to enter *alter-insert mode.* In alter-insert mode, the characters you type are added to the line immediately to the left of the character to which the cursor is pointing. Thus, we type precisely the letters "IOf " (the "I" could be lowercase).

```
00600        WRITE ('What number would you like the logs of? ');
00600        WRITE ('Of
                     ↑↑↑↑
                    IOf
```

Having reached the end of the text that we want to insert, we must signal EDIT that we have finished inserting characters. There are two ways to signal being done. Either ESC or RETURN may be used to leave alter-insert mode. The difference is that ESC returns you to *alter mode*, while RETURN returns you to EDIT *command mode.*

Since we are not yet done with this line, press the ESC key. There will be no visible indication of the ESC that you type.

- While in alter-insert mode, the character DELETE is effective for correcting typing errors. DELETE can also be used to delete characters to the left of the point where the I command was typed.
- CTRL/U in alter-insert mode does not erase the line. Rather, it restores the line to its original state (before the alter command), sets the cursor at the left margin, and returns to alter mode (rather than alter-insert mode).
- The LINE-FEED key in alter-insert mode will cause the altered line to be split, with the right end of the old line being placed on a new line immediately following the old one.
- The EDIT Extend (X) command works by altering the selected line, spacing to the right end, and then entering alter-insert mode. All the features of the alter command are available from the X command by pressing ESC to return to alter mode.

19.4.5. Alter Mode - Change One Character

We are again in alter mode pointing at the "W" in "What". We must change this to a lowercase "w". To do this, use the Change (C) command. After you type "C" in alter mode, EDIT waits for you to type another character. The character you type will be put into the line replacing the character that was there. The cursor will advance. The command to type is precisely "Cw" or "cw"; the case of the "C" is unimportant, but the "w" must be lowercase.

```
00600        WRITE ('Of w
                         ↑↑
                        Cw
```

Next, use the F command to find the "s" in "logs". Type "fs".

```
00600        WRITE ('Of what number would you like the log
                         ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
                        fs
```

19.4.6. Alter Mode - Delete Characters

We want to go past the "s", and then the rest of the line up to the question mark can be discarded.

To remove a single character, type the alter mode command D. Nothing will be visible. If you want to make sure that the character is gone, you might type P to print the line.

Try typing " DP":

```
00600        WRITE ('Of what number would you like the logsof? ');
                                                           ↑↑↑↑↑↑↑↑↑↑
                                                              DP
00600        WRITE ('Of what number would you like the logs
        ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
```

You can see that the space between "logs" and "of is no longer present. Now, you could at this point type "D" two more times, deleting the "of". You could even type "2D" which repeats the D command two times. However, it is clumsy to have to count, and it is difficult to see what you're doing. There is a better way.

19.4.7. Alter Mode - Kill Characters

To delete characters up to a specific character, use the alter mode Kill (K) command. Like the find command, the kill command must be followed by another character. The kill command will delete from the present pointer position up to, but *not including*, the next instance of the specified character.

We use the command "K?" to kill characters up to the question mark. As with the D command, K does not type anything. To find out what happened, you may type "P".[25]

```
00600       WRITE ('Of what number would you like the logs? ');
                                                        ††††††
                                                       K?P
00600       WRITE ('Of what number would you like the logs
           ††††††††††††††††††††††††††††††††††††††††††††††††††††
```

19.4.8. Alter Mode - Finish Line

When you finish changing the line, you can press RETURN key. EDIT types the balance of the line to the right of the cursor and returns to command level.

```
00600       WRITE ('Of what number would you like the logs? ');
                                                        ††††††
                                                       )
*
```

At this point we are finished with this line, and with our changes to LOGS.PGO. You may save it now, if you wish.

19.4.9. Alter Mode - Recovering from Blunders

If you discover yourself making a mistake while altering a line, there are several things you can do. First, by means of the commands described above, you can repair the damage.

Rather than retype the entire line, you may try CTRL/U. In alter mode, CTRL/U restores the line to its original condition, and places the cursor at the left end of the line. This is useful for starting the changes over again, if you have botched the command somehow.

A second command, "Q" means quit. The Q command is similar to CTRL/U in that it restores the line to its original form. However, Q also tells EDIT to return to command mode.

CTRL/U is also effective in alter-insert mode. But, the character Q, of course, is just a letter in alter-insert mode.

If you alter a range of lines, each time you finish with one line by pressing RETURN, EDIT goes on to alter the next in the range. If you want to return to EDIT command mode before finishing all the lines in the range, you may use the Q command. Note that no changes will be effected on the line where you type the Q.

19.4.10. Alter Mode - Summary of Commands

We have covered only a subset of the alter mode commands. Another useful command is "?" which types a summary of all the alter mode commands similar to what appears below.

In the command descriptions that follow, "n" means a numeric argument may precede the command. The "-" means

---

[25] If the alter mode K command cannot find the letter you specify, it does nothing. It will not kill to the end of the line.

that a minus sign may be present, in which case the command works *backwards*, where *backwards* usually means interchange the words "right" and "left", or "next" and "previous". Except where a statement to the contrary appears, "n" means the command is repeated "n" times.

-nSPACE	Move cursor to the right "n" places
-nDELETE	Move cursor to the left "n" places
-nFx	Move cursor right to the "nth" instance of the character "x"
-TAB	Move cursor to the right end of the line
-nW	Move cursor right to the beginning of the "nth" word following the original cursor position.
L	Print the line, return the cursor to the left margin.
P	Print the line, restore the cursor to its original position
RETURN	Finish making changes, print the balance of the line, return to EDIT command level (or continue to the next line in the range).
E	Same as RETURN, but omit printing the line.
nI	Enter alter-insert mode. Set inter-line increment to "n". To leave insert mode press ESC. RETURN leaves alter-insert and finishes the line. CTRL/U leaves alter-insert, restores the original line, and places the cursor at the left margin. LINE-FEED causes the present line to be split with the right end being placed on a new line following the present line. The line number of the new line is either the sum of the old number plus the current increment, or midway between the old line and next line following.
nC	Change the next "n" characters in the line to the next "n" characters typed after the "C".
-nD	Delete "n" characters to the right of the cursor.
-nKx	Delete characters to the right of the cursor up to, but not including, the "nth" instance of the character "x".
-nR	Delete "n" characters to the right of the cursor, then enter alter-insert mode.
CTRL/U	Restore the original line, set cursor to the left margin.
Q	Restore the original line and return to EDIT command mode.
X	Delete up to the beginning of the next word. Then enter alter-insert mode.
T	Transpose the next two characters.
nV	Invert the case of the next "n" characters.
^	Finish with line and alter previous line.
LINE-FEED	Finish with line and alter next line.
J	Split the line at this point, joining the right end to the front of the next line. No spaces are added.

19.5. Other Useful Commands

This section discusses pages in EDIT, and a variety of other useful commands.

19.5.1. Pages in EDIT

EDIT can divide a file into sections called pages. The pages are numbered consecutively, starting at one. When a paged file is printed, each page will start on a new piece of paper.

Each page has its own set of EDIT line numbers. Unless you specify otherwise, line numbers that you type are assumed to refer to the current page. Thus, if the current page is page 4, the line "3900" means "3900/4", i.e. line 3900 on page 4.

To reference a line on some other page, you must specify both the line number and the page number, such as "2300/2". The line number appears as usual, but a slash, "/", and a page number follow it.

You may use "." to signify the current page, and "*" to signify the last page. You can use "^" for the first page, but the first page is always numbered "1" which is easier to type than "^".

Sample commands:

R700/10	Replace line 700 on page 10.

`P/1:/3` Print pages 1 through 3.

`T4500,100/3:900`
Transfer lines 100 through 900 on page 3 to the current page, starting at (or after) line 4500.

Each field in a command is treated separately. Thus in the command "`T4500/3,100:900`" the "`100:900`" refers to the current page, not to page 3.

The second line number in a range specification pair is assumed to be on the same page as the first, unless you specify otherwise. For example, "`100/3:900`" is a shorthand for "`100/3:900/3`". In contrast, "`100:900/3`" means "`100/.:900/3`". which is a quite different thing.

Under some circumstances you may have to type "`./.`" to signify the current line. For example, if the current line is `500/3` then the command "`P100/1:.`" means, somewhat unnaturally, "`P100/1:500/1`".

The Mark command is used to create a new page. The command letter "`M`", followed by a line position, places a *page mark* just *before* the specified line. The specified line becomes the first line on the new page. The *page mark* is a special signal in the text of the file to mark where another page begins.

Because pages are numbered sequentially, the page number of each page following the new page is incremented by one.

To delete a page mark, type "`K/`" and the page number of the page mark you want deleted. Thus, to join pages 3 and 4, you would type "`K/4`"; you cannot delete page one. Naturally, the page numbers of all higher-numbered pages will be decremented by one. Frequently, this command will cause the "line numbers out of order" warning message. The command "`N,/.`" will correct the numbering.

You can leave EDIT and save your file without line numbers and without page marks by means of the `EK` command. `EK` is similar to `EU`, but `EU` preserves page marks.

19.5.2. Automatic File Saving - /SAVE: and /ISAVE:

There are two commands that cause EDIT to perform automatic file saving. Type these commands at the normal EDIT command level prompt.

`/SAVE: n` After every "n" commands that change the file, EDIT will execute a `B` command. Here, "n" means any number, e.g., "`/SAVE:20`"

`/ISAVE: n` Causes EDIT to execute a `B` command after every nth new line insertion.

19.5.3. Joining Two Lines - J Command

The J command joins two consecutive lines into one. For example, if lines 500 and 600 are consecutive as they appear in the example below, the command "`J500`" would join them into a new line 500. The new line contains the original contents of 500 at the left and the contents of 600 at the right. Notice that no spaces are added.

```
*p500:3
00500    This is line 500.
00600    This is the original 600.
00700    This is irrelevant
*j500
*p500:700
00500    This is line 500.This is the original 600.
00700    This is irrelevant
*
```

19.5.4. Justifying and Filling Text - JU Command

To *fill* a piece of text means to take the text word-by-word and fill each line with as many whole words as will fit between the margins.

To *justify* a piece of text means to make it line up evenly at a margin. You can justify text on two margins at once by adding extra spaces between the words to make each line start at the left margin and reach all the way across to the right margin. Adding spaces is called *padding*. The JU command fills and justifies text.

The following is an example of ragged text with a left margin of 1 and a right margin of 63:

```
*p2700:*
02700   To fill a piece of text means to take the text word-by-word and
02800   fill up each line with as many whole words as will
02900   fit between the margins.
03000
03100   To justify a piece of text means to make it line up evenly
03200   at a margin.  You can justify text on two margins
03300   at once by adding extra spaces
03400   between the words to make each line start
03500   at the left margin and reach all the way across to the right
03600   margin.  Adding spaces is called "padding".  The JU command
03700   fills and justifies text.
*
```

The same text follows. This time it is filled and justified, with a left margin of 10, and a right margin of 50. The paragraph is indented 5 spaces, yielding a paragraph margin of 15 (= 10+5). To say that the left margin is 10 means that column 10 is the first column used for text. Column 15 is the first column used for the first line of a paragraph.

The justify command to produce this transformation is "JU2700:3600,10,50,15". In this command, "JU" is the name of the justify command. The range of lines is "2700:3600". The "10", "50", and "15" are the left margin, right margin, and paragraph margin, respectively. The margins specifications, if present, should be separated by commas, as shown.

```
*ju2700:3600,10,50,15
*p2700:*
02700                To fill a piece of text means to
02800           take the text word-by-word and fill up
02900           each line with as many whole words as
03000           will fit between the margins.
03100
03200                To justify a piece of text means to
03300           make it line up evenly at a margin.  You
03400           can justify text on two margins at once
03500           by adding extra spaces between the words
03600           to make each line start at the left
03700           margin and reach all the way across to
03800           the right margin.  Adding spaces is
03900           called "padding".  The JU command fills
04000           and justifies text.
*
```

If you do not specify margins, the command normally uses 1 for the left margin, 69 for the right margin, and 1 for the paragraph margin. The justify command paragraphs whenever it finds a blank line. You should avoid the use of TAB characters in the text that is to be justified.

You may change the default margins by means of the commands "/LMAR:", "/RMAR:" and "/PMAR:". For example, "/PMAR:5" set the paragraph indentation to four spaces (i.e., start the text at column 5).

There are other EDIT commands related to JU:

JF	Fill without padding.
JC	Center each line in the range.
JR	Pad at the beginning of line to make an even right margin.
JL	Make an even left margin by adding or removing leading spaces.

19.5.5. Extended Character Set

The terminals that we customarily use permit the display of about 95 different characters. Some displays are not so flexible, and even with the 95 characters, some of the 128 possible characters have no graphic symbols at all.

Under some circumstances, you might find it necessary to enter one of these non-graphic characters into a file. There are two problems that must be overcome in using non-graphic characters: you must be able to type them in, and you must be able to see them when they are in your file.

If you must deal with these characters, EDIT has a mode in which the non-graphic characters can be typed in and typed out as a pair of characters. To enter this extended mode, the command `/C128` is used. In C128 (128-character) mode, the character apostrophe, ' (SHIFT/7 on most terminals), is used to modify the character that follows. Thus, a pair of characters starting with an apostrophe is considered to be one of these special characters. For example, in C128 mode, typing ' (while inserting text would result in the insertion of a delete into the text itself rather than either inserting the two characters "'" and "(" or performing the ordinary function of delete and deleting the previous character.

Some of these characters have special significance in the Find and Substitute commands. See page 128.

To revert to the normal mode, in which the apostrophe is not used as a prefix for a special character, use the EDIT command `/C64`.

The table of special characters and their codes appears in Appendix A, page 249.

19.5.6. Advanced Find and Substitute Features

The Find and Substitute commands have several more powerful features than any we have yet discussed. You should be relatively comfortable with the basic features of Find and Substitute before trying these.

The two advanced features of Find and Substitute are command switches and special matches.

Switches can be used to modify the function of a Find or Substitute command. If a switch is present in an F or S command, it should follow the line range. Each switch is a single letter preceded by a comma. Besides switches, a repeat count can be specified, also preceded by a comma.

A general example of a Find command with switches is

```
*Fstring$range,A,N,F,number                    $ means ESC
```

If the line range in the F command is omitted, the range is from the next line following the current line (that is, .+1) to the end of the file. If you desire to limit the range, only the second half of the range need be typed, e.g., `Fthis$:/10` will look for "this" from .+1 through the end of page 10.

The Find command switches have the following meanings:

A The A switch tells EDIT to enter alter mode on the line where the string is found. EDIT spaces forward in the line until the alter mode cursor is positioned at the first character of the search string. If a repeat count (number in the example above) is used with the A switch, then when you type carriage return to end the alteration, EDIT will continue the Find and Alter command. A continuation of the Find command (F without an argument) will leave the A switch in effect.

N The N switch tells EDIT to print only the line numbers and not the text of the lines where the string is found. This switch is really useful only if you have a slow terminal and an up-to-date listing of your file.

Use of the A switch overrides the N switch. As is the case with the A switch, the N switch remains in effect when a search is continued by an F command without arguments.

E Requires an exact match of upper and lower case letters. Normally, EDIT matches the search string

regardless of the case of the letters in the file. This switch forces EDIT to pay attention to the specific case of the characters in the search string, and locate a match only where both the file and the search are exactly the same.

number — Requests that EDIT find the specified number of occurrences of the search string. Each time one is found, the line containing it is typed (or just the number is typed if the N switch was used, or the line is altered if the A switch was used).

To see all lines containing the string "occur," one might use a command such as:

```
*Foccur$/^:/*,99
```

This presumes that there are less than 99 occurrences of "occur" in the file, and uses "99" only because it is easy to type.

Switches can be used in combination. For instance, to find the next line with the word "me" capitalized and then alter the line you could use the command:

```
*FMe$,A,F
```

In the Substitute command, switches may appear as they do in the Find command. The general form of the Substitute command is

```
*Sold-string$new-string$range,D,N,E,number   $ means ESC
```

The switches have the following meanings:

D — The D switch forces EDIT to allow you to decide whether each substitution is appropriate. EDIT will make the indicated substitution on a trial basis, and display the resulting line. EDIT then prompts by typing D*. Your response to that prompt may be any one of the following:

- SPACE — confirms the substitution for the line.
- DELETE — rejects the substitution for the line.
- A — enters Alter mode for the modified line.
- G — confirms the substitution for this line, and ends the effect of the D switch; if any further substitutions remain to be made in this range, they will happen without asking for a decision.
- E — rejects this substitution and returns to command mode without making further substitutions.
- H — types a help message explaining the responses to the D* prompt.

N — causes EDIT to print only the line number of the lines where it makes substitutions.

E — requires an exact upper and lower case match.

number — specifies the maximum number of lines to change by this substitution. When either the range is exhausted or this count runs out, the substitution process will stop.

For example, if you had capitalized all uses of the word "north" and subsequently realized that it should be capitalized only when used as a proper noun, you could use the following command:

```
*SNorth$north$,D,E
```

This would allow you to examine each substitution and determine whether it is necessary.

Sometimes the normal Substitute and Find commands don't provide enough flexibility for some difficult editing tasks. Using the special characters accessed by the C128 (128-character) mode, the Find and Substitute commands can be made more powerful. To use these special characters you must first give the EDIT command /C128. As soon as you have

finished using these characters, you should use the /C64 command to revert to the normal character set.

The first special character in Find and Substitute is "|", meaning, "match any separator". For example, to find "cat" but not "catatonic", you could use the command:

```
*FCAT|$                                    where $ is escape
```

This would search for the letters C A T followed by any separator. Note that words like "bobcat" or "ducat" are also found by this command.

Separators include all the punctuation, including end of line. However, the characters period, percent, and dollar sign are not normally considered to be separators. To force EDIT to treat period, percent, and dollar sign as separators, give the EDIT command /SEP. To revert to the default mode, give the command /NONSEP.

Suppose we have to change every instance of "car" to "automobile". We must proceed "carefully", avoiding unfortunate substitutions that would result in words such as "automobileefully".

We can write a Substitute command in which the old string (or search string) is written as |car|. This would look for the letters C A R surrounded on both sides by some separator. This would certainly avoid finding the letters C A R in the word "carefully". However, there is a problem in constructing the new replacement string. If the sequence "my car," appears, then the first separator is a space and the second separator is a comma. Whatever separator is found in front of "car" must be placed in front of "automobile"; whatever follows "car" must follow the replacement. In order to do this, we must first understand the concept of a *partially specified string.*

In a substitution, a partially specified string is that portion of the text in the file that matches one (or more) of these special search characters. In the search string |car| each of the vertical bars represents a partially specified string. In the case of the search string |car| matching the string "my car,", the first vertical bar in the search string corresponds to the space after "my", and the second vertical bar corresponds to the comma. The first vertical bar is the first partially specified string.

The partially specified strings in the match string can be included in the replacement string. The sequence '*1'* corresponds to the first partially specified string; the sequence '*n'*, where n is a number, is the n-th partially specified string. Thus, the substitute command that we want is

```
*S|car|$'*1'*automobile'*2'*$
```

The special character '/ matches any single character. For example the search string F'/'/D would find words like "find", "food", "arfvedsonite", "antifeudalism", etc.

The special character '% means do not match the next character. For example the search string FO'%OD would match "fold" or "ford" but it would not "food" because the third letter cannot be an "o". The sequence '%O means any character except "o". If you were doing a substitution, the sequence '%O in the search string counts as exactly one partially specified string.

Suppose that in a program you were writing you had used the variable name LCOUNT in some places and mistakenly called it COUNT elsewhere. Then, to change COUNT to LCOUNT you might try something like:

```
*S'%|count$'*1'*|count$%
```

Some combinations have special meanings. The combination '%'/ (not any character) matches either the beginning or end of a line, depending on context. Another character, '), means "any number of" the character that follows. For example, the string A')NY would match any of AY (zero N's), ANY, ANNY, ANNNY, etc.

These characters can be combined in interesting ways. For example, to delete all blank characters from the end of a line, the following substitution is possible:

```
*S'% ') '%'/$'*1'*$range
```

This example is worth noting carefully. In it, we search for the string composed of "anything except blank" followed by

"any number of blanks" followed by "end of line". The replacement string puts back the first partially specified string - the non-blank character; nothing else is present in the replacement string. The following is a summary of Special Search and Substitute Characters:

/C128 Representation	Matches
'/	Any single character.
|	Any single separator (does not include . % $).
'%	Not the next character.
')	Any number (including none) of whatever follows.
'7	Quote the next character.
'"	the next partially specified string.
'*n'*	the n-th (n is a number) partially specified string.

19.5.7. Using EDIT to Peruse a File - Readonly Mode

To view a file in READONLY mode, type the EXEC command "EDIT/READONLY", a space, the file specification of the file you wish to read, and RETURN.

```
@edit/readonly logs.pgo
Read: LOGS.PGO.6
*
```

When you are viewing a file in READONLY mode, you may use only the Print, Find, List and other *passive* commands. You cannot change the file.

The chief advantage of this mode is that it is faster than ordinary EDIT if you do not want to make changes. It also prevents you from making accidental changes. When EDIT isn't required to save anything, it can scan through your file much more quickly. This is an ideal way to read or print just a part of a file.

19.5.8. Writing Changes to a New File

After you start EDIT, if you decide that you want to keep the original version of the file, you can force EDIT to write the changes to a different file.

Any of the commands that write the current changes to a file will accept a new file name. These commands include E, EU, EK, G, GU, B, and BU. To specify a different file name, first type the command letter(s). Then, instead of pressing RETURN, press ESC and type the desired name for the output file; then press RETURN.

For example, the command "B$NULOGS.PGO", where "$" means ESC, would change the name of the output file to "NULOGS.PGO", save the file, and return to EDIT command level. Once you have specified a new output file name, it is remembered while you are in EDIT. That is, a subsequent "B" command will write a new generation of NULOGS.PGO.[26]

[26] Caution: if you exit from EDIT having written a file with a new name, you will have to mention the new name in the EXEC command EDIT when you want to make further changes to this new file. If you repeat the EDIT command without naming the new file specification, the EXEC will tell EDIT to make changes to the old file!

19.5.9. Getting Help from EDIT - H Command

The H command provides information about the other EDIT commands. To use the Help command, type "H" followed by the command you want information about. For instance "HT" will give you information about the Transfer command. The command "H?" will provide a list of commands which you can get help on.

19.6. EDIT Command Summary

The following is a complete list of EDIT commands (not including switches), with a brief description of each.

Items appearing within pointed brackets, "<" and ">" signify that some "syntactic" entity must be placed there. For example, "<range>" means a line range in any of standard forms. The pointed brackets do not actually appear in the command, they serve only to point out the syntactic elements.

A Alter - A<range> - Starts the Alter display editor.

B Backup - Saves your file with its EDIT line numbers and leaves you in EDIT command mode.

BI	Insert an identifying line before saving your file.
BJ	Delete the first line of the file and do a BI command.
BK	Save your file unnumbered, remove page marks.
BU	Save your file unnumbered.

C Copy - C<destination line>,<range> - Make a duplicate of part of your file and move it to another spot in your file.

Or - C<destination line>=<file specification>,<range> - Copy a specified part of another file into your present file.

Or - C<destination line>=<file specification>/s - Peruse another file, using P and F commands, then (optionally) copy part of that file into your present file.

D Delete - D<range> - Delete a range of lines from your file.

E Exit - E - Save your file with its EDIT line numbers and return to the EXEC.

Or - E<ESC><file specification> - Save your file with its EDIT line numbers and return to EXEC. If <ESC> <file specification> is given, save it under the given specification.

EI	Insert an identifying line before performing an E command.
EJ	Delete the first line of your file and do an EI.
EK	Save your file unnumbered, remove page marks, return to EXEC.
EQ	Don't save your file, just return to EXEC.
EU	Save your file unnumbered and return to EXEC.

F Find - F<search string><ESC><range>,<options> - Locates the first occurrence of the search string text within the specified range. If the range is omitted, it searches for the next occurrence of the text after the current position. If the search string is omitted, it uses the last string specified in an F command. Give the command "hf" to learn about the options.

G Go - G<ESC><file specification> - Perform an E, then do the last load-class command.

GI	Do an EI, then do the last load-class command.
GJ	Do an EJ, then do the last load-class command.
GK	Do an EK, then do the last load-class command.
GU	Do an EU, then do the last load-class command.

H Help

H?	List all available help topics.

H<topic> Type the help text for a specific topic. Usually, topics are command names. For example, HI gives information about the Insert command.

I Insert - I<position> - Begin line insertion at (or after) the specified line.

Or - I<position>,<increment> - Begin insertion at (or after) the specified line, using the specified increment.

Or - I<position>!<line-count> - Begin insertion at (or after) the specified position, using an increment sufficiently small to fit line-count new lines in.

J Join - J<position> - Join the specified line and its successor into one line. The line number of the second line vanishes.

JU Justify - JU<range>,<lmar>,<rmar>,<pmar> - Justify (by filling and padding) the text within the specified range, using the specified left, right and paragraph margins.

- JC Center - Center the text, without padding or filling.
- JF Fill - Do a JU without padding to the right margin.
- JL Left justify - Justify the text to the left margin, without padding or filling.
- JR Right justify - Justify the text to the right margin, without padding or filling.

K Kill - K/<page number> - Delete the specified page mark, joining two adjacent pages into one.

L List - L<range>,s - List the specified range of lines on the lineprinter. If the ",s" is included, don't print the EDIT line numbers; "suppress" them.

M Mark - M<position> - Put a page mark just before the specified line, making it the first on a new page.

N Renumber - N<increment>,<range>,<beginning line number> - Renumber the given range of lines by the specified increment, giving the first line the beginning line number.

- NA Add - NA<increment>,<range> - add the given increment to the specified range of lines.
- NP Pretend. Do an N, pretending the pagemarks aren't there. That is, at the end of each page, instead of setting the first line of the next page to 100 (or whatever), the NP command adds the increment to previous number and keeps going. If you number two pages using NP prior to killing the page mark between them, there will be no lines out-of-order message.

P Print - P<range><options> - Print the specified range of lines. To learn about the options, use the command "hp".

R Replace - R<range>,<increment> - Delete the given range of lines, and begin inserting at the first line, by the given increment.

S Substitute - S<search string><ESC><replacement string><ESC><range>,<options> - Replace every occurrence of the search string by the replacement string over the given range of lines. Use the command "hs" for further information.

T Transfer - T<destination line>,<range> - Transfer the specified range of lines to a spot beginning at (or after) the destination line.

V Invert case - V<range> - invert the case of the text throughout the given range.

- VV Same as V.
- VL Make all letters lowercase.
- VU Make all letters uppercase.

X	Extend - X<range> - Extend the right end of a line by entering Alter-Insert mode at the right end of each of the lines within the range.
?	Type the list of available commands.
LINE-FEED	Print the next line.
ESC	Print the previous line.
.	Move - .<line number> - Move the line pointer to the given line.
@	Indirect command file
=	Print switch setting - =<switch name> .
=?	Prints a list of all displayable switches.
/	Set switch - /<on-off-switch> - Set<on-off-switch> (turn it on). Or - /<value-switch>:<number> - Set<value-switch> to<number>. Or - /? - Print a list of all settable switches.

19.7. Additional EDIT Documentation

An *EDIT Reference Card* is available from the LOTS office for 25 cents.

20. Using EMACS

EMACS is a powerful, display-oriented text editor developed at MIT. It features an extremely flexible macro facility (TECO) for customization and extensibility, plus very complete internal documentation and help facilities.

EMACS is a display editor. That means that normally the text being edited is visible on the screen and is updated automatically as you type your commands. You see what you are editing rather than the commands you type to perform the editing.

20.1. Documentation

Since EMACS is a real-time display editor, it is rather difficult to describe it in published documentation. Also, the novice may find him or herself bewildered by the apparent complexity of the editor or by the need to memorize apparently hundreds of commands in order to use it effectively. This section attempts to introduce the reader to a few of the most basic EMACS concepts and commands -- just enough to allow you to get started using it, and to point you to further information when you decide you need it. Other documentation includes:

- An excellent tutorial on EMACS, available on-line. To use it, run the program "ETEACH":

  ```
  @eteach
  ```

  With this tutorial you learn basic EMACS commands by using EMACS itself on a specially designed file which describes commands, tells you when to try them, and then explains the results you see. It gives a much more vivid description than a printed manual.

- *EMACS Manual for TWENEX Users*, by Richard M. Stallman. MIT AI Memo 555; available for perusing in the manual racks or the bookshelves in CERAS, or from the Stanford Bookstore, the GSB facility, or the Computer Science Technical Reports office. This is the EMACS reference manual, and contains everything most users will ever need to know about EMACS. After you have finished using the ETEACH tutorial, you might be interested in browsing through this manual, even if your interest in EMACS is only casual.

- `EMACS:EMACS.DOC` and `EMACS:EMACS.INDEX` are files containing charts of all standard EMACS commands.

- Internal documentation. EMACS is largely self-documenting. Commands exist to describe what any given command or command character will do. The complete *EMACS Manual* is also readable from within EMACS, formatted in a fashion to make on-line search easy.

20.2. Running EMACS

Since EMACS is a display editor, it requires a terminal that can selectively change parts of the screen. This implies that it cannot be run on hardcopy and "unsophisticated" terminals. To use EMACS, then, you must be on a terminal with display capabilities. Contact your computer facility staff if you are unsure whether your terminal is sufficiently powerful.

Having logged in at an appropriate terminal, you should define the logical name `EDITOR:` to be EMACS; you can then use the `EDIT` command to modify your file. Note that defining `EDITOR:` is something you will have to do each time you log in. If you always use EMACS, you can include this definition in your `LOGIN.CMD` file which will be automatically executed each time you log in (see section 9.8.7). Note that on dial-in and Ethernet terminals you will have to be sure that your terminal type is set correctly. See section 10.1 on how to set your terminal type.

```
@define editor: sys:emacs.exe
@edit foobar.pas
```

Once you have started EMACS it will clear your screen and display the file you are editing. To exit EMACS, type `CTRL/X CTRL/Z` (two characters), or `CTRL/C`. To resume editing in EMACS after having exited, give the `EDIT` command with no filename. Don't give the command `EDIT FOOBAR.PAS`, as that forces EMACS to do a great deal of

initialization over unnecessarily.

20.3. BEMACS--Beginners EMACS

As you will soon see, EMACS is a very powerful editor. It has many, many commands. A beginner will learn some subset of these commands and build up to the more advanced features. One problem with EMACS is that you can accidently give a command by pressing a stray key. If you don't know what that command does, you could become very confused. For example, if you accidently press CTRL/W, some portion of your file will vanish. Experienced users know to use the CTRL/Y key to get it back, but beginners can wipe out large parts of their file this way. EMACS is full of land-mines.

In order to make life easier for the beginner, a novice library has been defined. The novice library has all of the simple commands and some of the intermediate commands that are considered safe (CTRL/W is not). This version of EMACS is called BEMACS. It works very simply. Every EMACS command is either enabled or disabled, depending upon whether it is in or not in the novice library. If you try to use one of the commands that are enabled, everything works the way it would in EMACS. If you try to use something that is disabled, it will tell you that the key is disabled. It will ask you whether you *really* want to do that command. If it was a stray key that you pressed, you can say not to execute the command. It will give you the option of playing around with the command, though. You can find out more information about it, try it just once, and even enable it if you choose to. If you enable a previously disabled key, BEMACS will remember it and let you use the command freely from that point on. Thus, you can customize BEMACS to do only the commands you understand.

There are two special things that you have to do to run BEMACS. First of all, you have to get a file called EMACS.PROFILE in your directory. This file is a list of E's and D's that stand for Enable and Disable (for the various commands). You can get this file into your directory by saying:

```
@copy emacs:emacs.profile
 <EMACS>EMACS.PROFILE.3 => EMACS.PROFILE.1 [OK]
@
```

Don't delete this file from your directory. This is where your preferences will be remembered (i.e., if you decide to enable some of the commands later on, your file will change but not anyone else's). If you eliminate this file, then BEMACS will do funny things.

The next thing you have to do is to define your editor to be BEMACS. Actually, the easiest way to do this is to add it to your LOGIN.CMD file. The LOGIN.CMD file is executed every time you log in (see section 9.8.7). These are the lines you need:

```
define editor: sys:bemacs
set program bemacs keep start
```

Add these two lines to your file (obviously, if you don't have a file, you need to create one). Once you make your LOGIN.CMD file correct, you can say

```
@take login
 End of LOGIN.CMD.1
@
```

For those of you who are truly beginners on this system and don't understand things like TAKE and LOGIN.CMD files, you give a single command that will do these two steps for you. You just say

```
@take emacs:bemacs
```

and it will carry out the steps outlined above and tell you what it is doing as it goes along. After this is done, you will be set up to use BEMACS.

20.4. Basic Editing with EMACS

This section discusses only the most basic features of editing with EMACS. You must know all the material presented here. However, to use EMACS effectively, you will find that you need to memorize more commands than is convenient in one reading or session. The ETEACH tutorial presents a few more; try it after you have read this section.

Text that you type while in EMACS is automatically inserted in the buffer as you type it; EMACS commands are generally one or two control characters which result in actions on the screen rather than being printed themselves. However, EMACS was originally designed for terminals slightly different from most of the ones found around Stanford. These terminals have two different control keys, called CONTROL and META (or sometimes CONTROL and EDIT). If your terminal does not have a META key, then you have to press the ESC key and then another key to get a META character. Rather than write out META or CONTROL each time we want to describe a command, we use the following abbreviations:

`C-<char>` means hold the CTRL key while typing the character <char> Thus, `C-F` would be what we have called elsewhere `CTRL/F`.[27]

`M-<char>` means press ESC, release it, then type the character <char>.

`C-M-<char>` means type `CTRL/Z` and then the character <char>.

Note an important and confusing asymmetry. `C-<char>` is typed by holding down two keys *simultaneously*, while `M-<char>` and `C-M-<char>` require that you type two keys *in sequence*.

Whenever you are working with EMACS, the cursor (the blinking underline or white blob) indicates the position at which editing takes place. Most of the commands we discuss here involve moving the cursor or deleting text around the cursor. The cursor is logically *between* two characters, and should be thought of as pointing immediately before the character that it appears on top of.

To insert printing characters into the text you are editing, just type them. They are inserted into the text at the cursor, and the cursor moves forward. Any characters on the line after the cursor move forward too. If the text in the buffer is FOOBAR, with the cursor before the B, then if you type XX you get FOOXXBAR, with the cursor still before the B.

To correct text you have just inserted, you can use DELETE or RUBOUT. The DELETE key deletes the character *before* the cursor (not the one that the cursor is on top of or under; that is the character *after* the cursor). The cursor and all characters after it move backwards. Therefore, if you type a printing character and then type DELETE, they cancel out. Note that BACKSPACE does not perform the same function in EMACS as it normally does on a Stanford DEC-20; use DELETE instead.

To end a line and start typing a new one, press RETURN. You can delete this line break with DELETE just as you can delete other characters; if you do so, the two lines are appended. If you add too many characters to one line, without breaking it with a RETURN, the line will grow to occupy two (or more) lines on the screen, with a "!" at the extreme right margin of all but the last of them. The "!" says that the following screen line is not really a distinct line in the file, but just a continuation of a line too long to fit the screen.

To do more than insert characters, you have to know how to move the cursor. There are several ways you can do this. One way (not always the best, but the most basic) is to use the commands Previous, Backward, Forward and Next. As you can imagine, these commands (which are given to EMACS as `C-P`, `C-B`, `C-F`, and `C-N` respectively) move the cursor from where it currently is to a new place in the given direction. Here, in a more graphical form are the commands:

---

[27] We are lying slightly here. EMACS actually makes a distinction between the meaning of MIT CONTROL key and the CTRL key on our keyboards, though most of the time you can use `CTRL/<char>` to get `C-<char>`, at least if <char> is a letter. Sophisticated users should see the *EMACS Manual* for more details.

```
                    Previous line, C-P
                            :
                            :
Backward, C-B .... Current cursor position .... Forward, C-F
                            :
                            :
                    Next line, C-N
```

Other commands necessary for moving around in a file include C-V, which moves to the next screenful of information, M-V, which moves to the previous screenful, and C-L, which clears and redisplays the screen.

In addition to inserting text and moving around, you need to know how to delete text and correct typing errors. You can delete the character immediately following the cursor by typing C-D or the character immediately before the cursor with DEL. To delete everything up to the end of the line (or to combine two lines, if you are already at the end), type C-K.

To swap two characters, type C-T. Transposing two characters as you type is the most common typing error; if you develop a habit of catching these errors and immediately typing C-T, you will be amazed that you were ever able to live without this correction feature.

To exit EMACS, type C-X C-Z (two characters). Note, though, that EMACS works on a "buffer" that contains a copy of your file, and does not change your file itself unless you explicity instruct it to do so. To save your file, first type C-X C-S before exiting.[28] You can also use the C-X C-S command as a "backup" command to save your work during your editing session, to prevent loss of data if the system crashes.

To summarize, among the "basic set" of essential EMACS commands are:

C-B	Backward one character (cursor left).
C-F	Forward one character (cursor right).
C-P	Previous line (cursor up).
C-N	Next line (cursor down).
C-V	Move to the next screenful of information in the file.
M-V	Move to the previous screenful of information in the file.
C-L	Clear the screen and redisplay everything (useful if your screen has been messed up by an asynchronous message).
C-D	Delete next character.
DEL	The DEL or RUBOUT key deletes the previous character.
C-T	Swap two characters.
C-K	Delete to end of line.
C-X C-S	Save your file.
C-X C-Z	Exit from EMACS without saving the file.

Simple commands also exist for moving over or deleting whole words, sentences, and paragraphs, for changing capitalization, for moving and copying blocks of text, for editing several files simultaneously, and for performing almost

---

[28] C-X C-S actually makes a new version of your file, deleting but not expunging the old one. So if you find yourself having saved disastrously bad changes to your file, try UNDELETE to recover a better version.

If you run out of disk space while trying to save your file, EMACS will give an error message "?Quota exceeded or disk full, and will return you to the EXEC leaving your screen a mess. Type BLANK to clear your screen, obtain more space (perhaps by EXPUNGE), then give a START command to restart EMACS so you can save your work. If the message is immediately repeated, ask for help.

any other changes. In addition, you can customize EMACS's behavior; for example, if you want EMACS to automatically start a new line when you approach the right margin you can set "Auto Fill Mode". Run the ETEACH tutorial for more information, then consult the *EMACS Reference Manual.*

20.5. Common Pitfalls

If you are an inexperienced user with EMACS, you should probably be using BEMACS (see section 20.3), but for those who run into trouble with EMACS, here is some advice. Particular problems for the novice are special modes, subsystems, and subcommands. For example, if you type `ESC ESC` (an easy thing to do by accident, since you will often want to type a single `ESC` to create a `M-<char>` command), you get into something known as a "minibuffer" which provides the experienced user with access to some advanced EMACS features. You can get out of most such situations by typing one or two `C-G` characters.

Some EMACS commands can have apparently drastic effects on your file. For example, the command `C-W` (Kill Region) deletes all text between the cursor and the "mark" (sort of a second invisible cursor which you can move around by various commands). If you accidentally type `C-W`, you may see a large part of your file deleted. However, all is not lost, since there is an "Undo" command which, if given immediately after a disaster, will often successfully fix your file. To give the Undo command, type `M-X undo` followed by a carriage return.[29] The Undo command will successfully fix many drastic changes to the buffer, not just an accidental `C-W`.

`C-G` Abort whatever EMACS is doing and get back to reading keyboard commands.

`M-X undo` Undo the last major change or deletion.

20.6. On-Line Help

One EMACS command is particularly helpful for obtaining additional documentation. `C-_` (that is, control underscore, the underscore being the shift of the dash on most terminals) is the "HELP" key.[30]

To use the HELP character, type `C-_` and another character saying what kind of help you want. If you are *really* lost, type `C-_ ?` and EMACS will tell you what kinds of help it can give. If you have typed the help character and decide you don't want any help, just type `C-G` to abort. The most basic HELP feature is `C-_ C`. Type "`C-_`", a "`C`", and a command character, and EMACS prints a description of the command. When you are reading it, type a space or a `C-G` (quit) to bring your text back on the screen.

`C-_ C` Describe the effect of an EMACS command (keystroke).

`C-_ I` Run the INFO document reader. INFO gives you access in a structured fashion to the whole *EMACS Manual*, in addition to documentation on how to use INFO, and on various other topics. Try it; if you don't like it, you can get back to EMACS by typing "`Q`".

`C-_ A` List the names of all functions which contain a word you specify. For example, to find out what commands deal with sentences, you might say `C-_ A sentence`.

---

[29] This is an example of an "extended" Emacs command, one with a name rather than just a command character. The `M-X` is a sort of command prefix that informs Emacs you are about to type a long-form named command.

[30] On some terminals it is not possible to type `C-_` by holding down `CTRL` and typing `_`. On VT100s, the "HELP" key is typed by typing `CTRL /?` and on Heath or Zenith terminals, it is just `CTRL /_`.

20.7. Intermediate EMACS

EMACS has commands for doing just about any editing task you can imagine. In addition, if you do find a task for which there is no particular command, you can write a command of your own. For a discussion of customization and extension, consult the *EMACS Manual.* Furthermore, EMACS has various "modes" of operation in which some commands have slightly changed meaning. For example, in Pascal Mode (useful for editing Pascal programs), the `LINE-FEED` character is redefined to end the current line then indent the next line to a place appropriate for structured programming. Normally, the mode is selected for you automatically based on the file type, but you can change it, just as you can change almost everything about the way EMACS behaves.

The following is a command summary, organized by topic, of the default EMACS command environment in "Fundamental Mode":[31]

Cursor Control

Char	Control Bits	Meaning
	C-A	Beginning of Line
A	M-A	Beginning of Sentence
	C-M-A	Beginning of DEFUN
	C-B	Backward Character
B	M-B	Backward Word
	C-M-B	Backward S-Expression
D	C-M-D	Down List
	C-E	End of Line
E	M-E	End of Sentence
	C-M-E	End of DEFUN
	C-F	Forward Character
F	M-F	Forward Word
	C-M-F	Forward S-Expression
H	C-H	Backward Character
M	M-M	Move to beginning of text
	C-M-M	on current line
	M-[	Move to beginning of paragraph
[	C-M-[	Move to beginning of DEFUN
	C-X [	Move to beginning of page
	M-]	Move to end of paragraph
]	C-M-]	Move to end of DEFUN
	C-X]	Move to end of page
	C-N	Move to Next Line
N	M-N	Move to Next Comment Line
	C-M-N	Move across Next List
	C-P	Move to Previous Line
P	M-P	Move to Previous Comment Line
	C-M-P	Move across Previous List
U	C-M-U	Up List
V	C-V	View Next Page
	M-V	View Previous Page
X	C-X ↑X	Exchange point (cursor) and mark

Display

Char	Control Bits	Meaning
L	Form or C-L	Clear and redisplay screen
R	M-R	Move cursor to screen line
	C-M-R	Reposition window

Searching

Char	Control Bits	Meaning
R	C-R	Reverse Search
S	C-S	Search (forward)

Moving Text

Char	Control Bits	Meaning
	C-T	Transpose Characters
T	M-T	Transpose Words
	C-M-T	Transpose S-Expressions
	C-X tT	Transpose Lines
	C-X T	Transpose Arbitrary Regions

Prefix Characters

Char	Control Bits	Meaning
Escape	Escape	M- Prefix Character (doesn't do this in a mini-buffer)
\	C-\	M- Prefix Character (good for use in mini-buffer)
Q	C-Q	Quote Prefix (Insert next chr typed)
U	C-U	'Universal' Argument Prefix
X	C-X	C-X Command Prefix

[31] Beware. Since you can customize EMACS to suit your fancy, this description may not be completely accurate. Also, EMACS may have changed since this description was written.

Z	C-Z	C-M- Prefix Character
↑	C-↑	C- Prefix Character

Reading and Writing Files

Char	Control Bits	Meaning
F	C-X ↑F	Find File
Q	C-X ↑Q	Inhibit Auto-Writeback
R	C-X ↑R	Read File
S	C-X ↑S	Save File (under default filename)
V	C-X ↑V	Visit File
W	C-X ↑W	Write File (reads a filename)

Buffer Manipulation

Char	Control Bits	Meaning
A	C-X A	Append to Buffer
B	C-X B	Select Buffer
	C-X ↑B	List Buffers
K	C-X K	Kill Buffer

Indentation

Char	Control Bits	Meaning
	Tab or C-I	Indent According to Mode
I	M-I	Indent to Tab Stop
	C-M-I	Indent for Lisp
	↑X Tab	Indent Rigidly
J	Linefeed or C-J	Indent New Line
Q	C-M-Q	Indent All Lines Next S-Expression
;	M-;	Indent for Comment

Comment Fields

Char	Control Bits	Meaning
J	M-J	Indent New Comment Line
	C-M-J	Indent New Comment Line
N	M-N	Move to Next Comment Line
P	M-P	Move to Previous Comment Line
	M-;	Indent for Comment
;	C-M-;	Kill Comment
	C-X ;	Set Comment Column

Marking a Region

Char	Control Bits	Meaning
	M-H	Mark Paragraph
H	C-M-H	Mark S-Expression
	C-X H	Mark Whole Buffer
P	C-X ↑P	Mark Page

Text Deletion

Char	Control Bits	Meaning
	Rub	Delete character backward
Rubout	M-Rub	Delete word backward
or	C-M-	Delete S-Expression backward
Delete	Rub	
	C-X	Delete sentence backward
	Rub	
	C-D	Delete character forward
	M-D	Delete word forward
	C-K	Kill Line Forward
K	M-K	Kill Current Sentence
	C-M-K	Kill S-Expression Forward
W	C-W	Kill Region
;	C-M-;	Kill Comment

Text Copying

Char	Control Bits	Meaning
A	C-X A	Append Region to Specified Buffer
	M-W	Copy Region
W	C-M-W	Make next kill join previous in kill buffer

Restoring Deleted/Copied Text

Char	Control Bits	Meaning
	C-Y	Yank item from killring to buffer
Y	M-Y	Yank "the kill before that"
	C-M-Y	Mini-buffer only: Yank last mini-buffer command string
		RMAIL Reply: Yank text of mess.
		(Not defined anywhere else)

Blank Lines/Carriage Returns

Char	Control Bits	Meaning
	Return	Insert (CR LF)
	C-O	Open a Blank Line

O	C-M-O	Split Line; move text vert. down
	C-X ↑O	Kill Blank Lines after this line end

Subsystems

Char	Control Bits	Meaning
D	C-X D	DIRED (a Directory Editor)
I	C-X I	INFO (Documentation Reader)
M	C-X M	MAIL (send MAIL to other users)
R	C-X R	Read MAIL (using MM)

Teco Related Commands

Char	Control Bits	Meaning
G	C-X G	Get from Q-Register
X	C-X X	Put in Q-Register

Extended Commands

Char	Control Bits	Meaning
X	M-X	Read Extended command and args, then execute
	C-M-X	Execute Extended command

Quit/Exit Commands

Char	Control Bits	Meaning
G	C-G	Abort currently executing command
Z	C-M-Z	Exit current editing level
	C-X ↑Z	Exit EMACS - Return to Superior
]	C-]	Abort a recursive ↑R-mode (eg flush Mail Reply without sending)

Setting Column Variables

Char	Control Bits	Meaning
F	C-X F	Set Fill Column (Fill commands)
N	C-X ↑N	Set Goal Column (↑N and ↑P)
;	C-X ;	Set Comment Column

Directory Hacking Commands

Char	Control Bits	Meaning
D	C-X D	DIRED (a Directory Editor)
	C-X ↑D	Partial directory listing

Changing Case

Char	Control Bits	Meaning
C	M-C	Capitalize word
L	M-L	Lowercase word
	C-X ↑L	Lowercase Region
U	M-U	Uppercase word
	C-X ↑U	Uppercase region

Formatting Text

Char	Control Bits	Meaning
F	C-X F	Set Fill Column
G	M-G	Fill Text Region
	C-M-G	Grind Lisp or Macsyma code
Q	M-Q	Fill paragraph
S	M-S	Center text on line

Information re Buffered Text

Char	Control Bits	Meaning
L	C-X L	Count Lines on Current Page
=	C-X =	Info re Screen and Buffer Position

Dealing with 2 Windows

Char	Control Bits	Meaning
1	C-X 1	Return to 1-window mode
2	C-X 2	Enter 2-window mode
3	C-X 3	Enter 2-window mode (stay in 1)
4	C-X 4	Enter 2-window mode with options
O	C-X O	Move to Other Window
V	C-M-V	View Next Page of other window
↑	C-X ↑	Grow Window

Changing Virtual Buffer Bounds

Char	Control Bits	Meaning
N	C-X N	Narrow Bounds to current Region
P	C-X P	Narrow Bounds to current Page
W	C-X W	Widen Bounds (to whole buffer)

21. Using FORTRAN

FORTRAN (FORmula TRANslation) is perhaps the oldest computer language that is still in widespread use today. It is especially popular among scientists and engineers, and is particularly suited for numeric calculations. It is perhaps the most widely available programming language, and hence is well suited for writing portable programs. It is also the only widely available language which supports double precision calculations.

21.1. Documentation

This chapter contains hints on how to write, run, and debug a FORTRAN program. It is oriented towards people who already have some experience with FORTRAN (perhaps on an IBM computer) and need to know about idiosyncracies of the DEC version of the language.

The language reference manual for FORTRAN is *LOTS Fortran*, which includes the DEC *Fortran-20 Reference Manual* [AA-4158B-TM], as well as a version of this section. It is available in the Stanford Bookstore.

Users without previous experience programming in FORTRAN may wish to consult one of the standard textbooks on the subject. Numerous excellent textbooks on FORTRAN programming exist. For experienced FORTRAN programmers, one interesting book is *Fortran Techniques* by A. C. Day, which contains advanced techniques and hints on writing portable and readable FORTRAN programs.

21.2. Creating your Program

The first step in running a FORTRAN program is to place the source of your program in a file. To do so, you would normally use a text editor such as EDIT or EMACS (via the system commands `CREATE` and `EDIT`) to enter and correct the text of the program. If your program is stored on another system (such as the I.T.S. IBM computers), information on transferring it by magnetic tape may be obtained by typing "`HELP TAPES`" to the EXEC. If your program is on a Stanford computer on the Stanford Ethernet, you can use the FTP program to transfer it to your DEC-20. In any case, the source file corresponds to the deck of cards that might be read through a card reader on other systems.

If you are entering your program at a terminal, note that you may use the `TAB (CTRL/I)` key rather than spacing to column 6 or 7. An initial tab followed by a letter is interpreted as spacing to column 7, and hence as the beginning of a new statement. A tab followed by a digit (1 to 9) is interpreted as spacing to column 6 even though it appears in column 9 of your listing, and hence as a continuation line. See the *LOTS Fortran Manual*, chapter 2, pp. 2-3 for further details.

21.3. Files Involved in Running FORTRAN

To run a FORTRAN program, you must first compile the source into relocatable object code, then load this object code plus any external routines into your core image, and then either begin execution or save a copy of the core image for later execution. Each of these three steps requires the running of a different program, analogous to the job steps in an IBM batch job. There are EXEC commands that perform various combinations of these steps; these commands correspond to standard varieties of catalogued procedures available on the IBM machine, except they are available with a single command.

File specifications for FORTRAN consist of a file name, a period, a file type, plus optional fields to refer to files belonging to other users. FORTRAN requires that the file name for a program contain six characters or less, and that the file type be three characters or less. It is standard practice for the file type to designate its place in the chain of files from source to load module and data. The standard file types are:

Type	Meaning
FOR	FORTRAN language source file
REL	relocatable (object) module
EXE	executable program (load module)
DAT	data file

Thus a typical source file might be named MACHMN.FOR, and its associated relocatable binary file, MACHMN.REL.

21.4. Running a FORTRAN Program

The commands used to run FORTRAN programs are COMPILE, LOAD, EXECUTE, START, and DEBUG. Most of the time, though, you will only need to use EXECUTE. The EXECUTE command will take a source file, compile, load, and start it. For example, consider the following terminal session:

```
@create machmn.for
Input: MACHMN.FOR.1
00100    C Compute machine unit roundoff from definition
00200          double precision fuzz,ofuzz
00300          fuzz=1.0d0
00400    10    ofuzz=fuzz
00500          fuzz=fuzz/2.0d0
00600          if (1.0d0 + fuzz .gt. 1.0d0) goto 10
00700          write (5,*) ofuzz
00800          stop
00900          end
01000    $     <--- Press ESC, not "$" here
*e

[MACHMN.FOR.1]
@execute machmn
FORTRAN: MACHMN
MAIN.
LINK:  loading
[LNKXCT MACHMN Execution]
 0.2168404344971008870-18
STOP
END OF EXECUTION
CPU time: 0.06 Elapsed time: 3.31
@                                        all done
```

Since no explicit file type was given, the EXECUTE command searched for a file with one of the standard source file types, found a .FOR, and therefore called FORTRAN. You could have optionally typed the file name including the .FOR file type. (The "." must be left off or the entire .FOR must be included. A "." alone indicates a null file type.) When execution has finished, you will be returned to the EXEC.

The LOAD command compiles the source file (if necessary) and loads it into memory, but does not start execution. The EXEC's START command may be issued to do this. Thus LOAD plus START is equivalent to EXECUTE. The COMPILE command merely compiles a REL file. The DEBUG command is explained below.

Some optimization is done in each of the above commands. If a REL exists that is more recent than the source (FOR) file, then a new REL file is not created. This includes the COMPILE command, which then does nothing. To force compilation in such cases, you can include a /COMPILE switch.

A cross-reference listing may be generated via the /CREF switch, which is typed with a slash as below. However, if a recent REL file exists, no compilation will be done and hence no listing will be produced. You must force a compilation by including a /COMPILE switch as well:

```
@compile /cref/compile machmn
```

21.5. Debugging your FORTRAN Program

The most common cause of obscure errors in FORTRAN programs on the DEC-20 is problems with subroutine linkage. Make sure that the arguments in calls to subroutines agree in number, in position, and in type with the declared parameters of the routines.

Note that DEC FORTRAN passes *all* parameters, including constants, by reference. It is possible to destroy the value of a constant by passing it to a subroutine which changes the values of its parameters. The following program prints `J = 1`, not `J = 1000000`:

```
      CALL MESS(1000000)
      J=1000000
      WRITE(6,4000) J
4000  FORMAT(' J = ',I10)
      END
C
      SUBROUTINE MESS(I)
      I=1
      END
```

FORTRAN programs occasionally produce obscure error messages. Some of the more common error messages, and their usual causes, are:

`%FRSAPR Divide check`
: Attempt to divide by zero.

`? PA1050: ILLEGAL INSTRUCTION`
: You probably indexed outside the bounds of an array, and destroyed part of your program. Probable cause: wrong number of parameters to a subroutine, or incorrect array bounds in a subroutine parameter. The bounds declared in the subroutine should usually match those declared in the calling routine.

`%FRSAPR Integer overflow`
: Attempt to compute an integer greater than 2**35-1. Often, passing a REAL value to a routine that expects an INTEGER (e.g. reading into an INTEGER variable using an F format).

`%FRSAPR Floating underflow`
: Often, either an attempt to compute a REAL number that is very close to zero (absolute value less than .14E-38, e.g. by multiplying two small numbers), or passing an INTEGER value to a routine that expects a REAL (e.g. writing an INTEGER variable with an F format). If the underflow occurred in a numeric calculation, then it may not be a problem; the result is taken as zero, which might be what you intended.

`%FRSAPR Floating overflow`
: Attempt to compute a REAL greater than 1.7E38.

`%LNKSA No start address`
: No main program found. Perhaps the last time you compiled you had an error, but you gave the EXECUTE command again without changing anything, so the EXEC didn't bother to recompile, and used the old, erroneous, `REL` file. Fix: say `EXECUTE/COMPILE` to force recompilation and see your errors.

`%FRSOPN File was not found`
: Normally, this message indicates an attempt to read from a nonexistent file. Check the file name specified in the error message to make sure that it exists. Alternatively, you may be trying to read or write to a file whose protection does not allow you to access it.

`%LNKMDS Multiply-defined global symbol MAIN.`
: Two main programs found. You probably have an extra END statement in one of your routines. If the symbol is anything else except MAIN, it probably indicates that you have included a subroutine with that name twice.

`% LNKUGS Undefined global symbol XXX`

The specified symbol (XXX) was not declared. If XXX is the name of one of your subroutines then you forgot to write it. You may have omitted mentioning the name of the file or library file that contains this routine. If you don't recognize XXX, it may be a spelling error. Search your program for the string XXX.

If you get a divide check, overflow, or underflow, you can often find out where in your program it is occurring by using the `EXAMINE` command in the EXEC. Suppose you got an error message which mentioned a PC (program counter) of 1157. `EXAMINE` the PC value that was reported. The EXEC will type the symbolic name of the location you specified and its contents. The label typed will probably give you a clue to where in your source program the problem occurred.

```
%FRSAPR Floating divide check   PC= 1157

END OF EXECUTION
CPU time: 0.65  Elapsed time: 3.10
@set typeout mode symbolic
@examine 1157
500L+3/ FDVR 2,FDDT.+2
@
```

If the EXEC prints a number followed by "L", then the error occurred at that EDIT line number (in the example, `500L+3` means the fourth machine instruction generated for the statement beginning at line 500). If the EXEC prints a number followed by "P" (e.g. `10P+5`), then the error occurred in a statement whose FORTRAN statement number was that number (thus, `10P` means a statement labeled 10).

If the EXEC prints a number followed by "M", then it is more difficult to find out the source line where the error occurred. If you find yourself in this situation, use the `EXAMINE` command to examine locations preceding the one in which the error occurred. In the example above, you might try `EXAMINE 1150`. You may repeat the `EXAMINE` command until a label appears that has either a "P" or an "L" in it. The offending instruction appears at or after the line you identify by this procedure.

A powerful source-level dynamic debugger called FORDDT exists for FORTRAN. The `DEBUG` command behaves exactly like the `EXECUTE` command, except that it starts in debug mode, where FORDDT is available to help you interactively control execution of your program. You may examine and change variables, execute statements one at a time, insert breakpoints so that the program will stop each time it reaches a particular point, display status information, etc. FORDDT is documented in the *LOTS Fortran Manual*, Appendix E. To debug a program, type:

```
@debug machmn
```

21.6. Turning a Program into an Executable File

Once a program is running correctly, you might use the EXEC `SAVE` command to create a runnable file for repeated use. Note that at this stage you probably no longer need debugging information; by recompiling with the `NODEBUG` switch you will increase the execution speed of your program substantially. Even further savings can be obtained by also optimizing your program by means of the `OPTIMIZE` switch.[32] For example,

```
@load machmn/nodebug/optimize/compile
FORTRAN: MACHMN
MAIN.
LINK:  loading
EXIT
@save
 MACHMN.EXE.1 Saved
```

Now you have a file that can be run directly, without recompiling or loading, merely by typing its name to the EXEC. For

[32] However, optimization is no substitute for an efficient algorithm. At best, optimization will increase the speed of your program by a factor of 3 or 4; a program that takes an hour of cpu time to run unoptimized will still take much too long even after optimization. In such cases you should rewrite your program, or consider moving your program to a faster computer.

instance, you might sometime later type:

```
@machmn
 0.216840434497100887D-18,
STOP
END OF EXECUTION
CPU Time: 0.04 Elapsed time: 2.35
@
```

21.7. FORTRAN Data Handling

On the DEC-20, data is always input from a data file or terminal rather than cards. If you are accustomed to using cards, it may be helpful to think of each line of the input file as representing one card. Do not use tabs or CTRL/I when preparing the input file. Output may be to a file, to the terminal, or to the line printer.

21.7.1. Unit Numbers

If your program reads or writes data, it will use the standard FORTRAN unit designators to distinguish input and output of various kinds. As is usual, some unit numbers have default device assignments and some do not, but all may be defined or changed. The default assignments of the most commonly used and their meanings are:

FORTRAN unit no.	default filename	logical name	meaning
1	FOR01.DAT	DSK:	disk file
3	FOR03.DAT	LPT:	line printer (output only)
5	FOR05.DAT	TTY:	your terminal
6	FOR06.DAT	TTY:	your terminal
20	FOR20.DAT	DSK:	disk file
21	FOR21.DAT	DSK:	disk file
22	FOR22.DAT	DSK:	disk file
23	FOR23.DAT	DSK:	disk file
24	FOR24.DAT	DSK:	disk file

The association between a unit number and a device is defined via a logical name. These are always terminated by a colon, as you can see from the table of defaults above. When the unit corresponds to disk (DSK:), a file of the name indicated will be read or written. It is possible to override this default device or file name in an OPEN statement (see chapter 12.2 in the FORTRAN manual) or a CALL DEFINE FILE (which refers to an intrinsic subroutine providing compatibility with IBM FORTRAN; see page 15-17 of the FORTRAN manual). Many times the default name is adequate.

In order to change the default definition of one of the above unit numbers, or to use one which is not listed, the EXEC command DEFINE is used prior to execution of the program (you might compare this with a //FTnnF001 DD card in an IBM batch job). For example, to have input or output to unit 10 use the file MYNEWDATA.DAT, you would use a command such as:

```
@define 10: mynewdata.dat
```

Here, "10" is the unit number you wish to have correspond to a disk file. To return a unit number to its default, issue the DEFINE command without a defining logical name, as in:

```
@define 5:
```

Many IBM FORTRAN programs, such as programs imported from I.T.S., will work best with unit 5 (IBM 370 batch job stream) defined as DSK: to read from a file (by default FOR05.DAT), rather than its default of TTY:. Unit 6 (IBM 370 printer output) is also defaulted to TTY:, so if you don't want your output to go to the terminal, you must redefine 6:

accordingly.

21.7.2. Carriage Control Characters

If a file is written with carriage control characters, they will be obeyed if the unit is TTY: or LPT:. A "1" in the first column of the output line will be interpreted as meaning "start a new page" instead of being printed as itself; a space will be taken as "print on the next line", and so on. If the file is written to disk, the carriage control characters are preserved in the file. To make them work when sending the file to the printer, use the PRINT command with the /FILE:FORTRAN switch, e.g.,

```
@print for23.dat/file:FORTRAN
```

Be very careful not to specify the /FILE:FORTRAN switch for files that do not contain carriage control characters, e.g. input data files, or files which have EDIT line numbers. Note also that neither terminals nor the Printronix line printer can overstrike most characters, so a "+" in column one is generally useless.

Carriage control characters recognized by standard FORTRAN are summarized in the table below. DEC FORTRAN recognizes several additional carriage control characters, but their use is not recommended in programs designed for portability.

(blank)	single space
0	double space
1	go to new page
-	triple space
+	overprint

21.8. Incompatibilities with Other Versions of FORTRAN

DEC FORTRAN attempts to conform to ANSI standards for FORTRAN-77. Problems that arise generally are due to non-standard extensions in either another FORTRAN for which code was written, or in the DEC FORTRAN. For instance many features of WATFIV are not present, or are different (for example the PRINT statement). In DEC FORTRAN, dollar signs ($) are not legal in variable names, data initialization cannot take place in type specification statements, etc. On the other hand, DEC FORTRAN also has numerous non-standard extensions, including treating lower case as equivalent to upper in keywords and identifiers, multiple statements on a line (separated by ";"), use of "!" as a comment character, OPEN, TYPE, ACCEPT, REREAD, ENCODE, and DECODE statements, etc.; see the Reference Manual for details.

In other instances, difficulty arises where hardware differences render compatibility impossible. For example, random number generators often depend on knowledge of the word size of the machine. Note also that real numbers on the DEC processor have a range of from .14*10**-38 to 1.7*10**38, compared with the IBM 370's range of 7.2*10**75 to 5.4*10**-79. If the /GFLOATING compiler switch is specified, however, the range of real numbers on the DEC processor is 2.78**-309 to 8.99**307, which far surpasses the IBM 370's floating point range.

Another source of incompatibility arises from the fact that the FORTRAN at Stanford by default compiles with the DEBUG:ALL switch when you use any of the FEXEC compilation commands. Therefore, all array references are checked to insure that the subscript is inside the declared bounds of the array. This allows numerous programming errors to be caught, but it makes illegal the common FORTRAN coding trick of dimensioning subroutine parameters to 1 if the true dimension is unknown:

```
      function vecsum(vector,nvec)
      dimension vector(1)
      vecsum=0.0
      do 10 i=1,nvec
10       vecsum=vecsum+vector(1)
      return
      end
```

To fix the problem, and give FORTRAN as much chance as possible to catch other errors, replace the dimension in VECTOR(1) with the name of a parameter containing the true dimension:

```
dimension vector(nvec)
```

If you have a large existing program that uses this unfortunate coding trick, you can still use the program by compiling it with the /NODEBUG switch.

Other assorted differences between IBM FORTRAN-IV and DEC FORTRAN are discussed in the file DOC:FORTRAN.DIF.

21.9. FORTRAN-77

In the middle of 1983, a new version of DEC FORTRAN, version 7, was installed at Stanford. FORTRAN Version 7 is based on the standard American National Standard Programming Language FORTRAN - ANSI X3.9-1978 (FORTRAN-77). Copies of this standard may be obtained from:

American National Standards Institute
1430 Broadway
New York, N.Y. 10018.

Because of incompatibilities between versions 6 and 7 of DEC FORTRAN, a number of numerical analysis libraries such as IMSL, NAG, and EDA will not work with version 7. Until such time as the vendors of these packages upgrade their software to work with version 7, version 6 will be the default version of FORTRAN at most Stanford sites. Consult the help files on your system for further information on how to use FORTRAN-77.

There are numerous enhancements to Version 7 from Version 6:

- Character data at the FORTRAN-77 full language level. Supported features include character assignments, character relationals, substrings, concatenation, and character functions and arguments, including functions and dummy arguments of length "*". Character data is supported in DATA, COMMON, and EQUIVALENCE statements, and in formatted, binary, and image mode I/O. Refer to the DEC-20 FORTRAN manual for more information.

- IF-THEN-ELSE and END-IF statements.

- DO-WHILE and END-DO statements.

- Expressions on output lists (you may use "TYPE 10, A+B", for example).

- Intrinsic and generic functions at the FORTRAN-77 full language level. Intrinsic and generic functions are those that are pre-defined by the FORTRAN system; previously, there was not full support of all standard functions.

- Single-record and multi-record internal files. Internal files provide the ability to perform formatted data transfers between character variables and the elements of an I/O list. This is very similar to the ENCODE/DECODE facility.

- FORTRAN-77 DO loop semantics (DO WHILE, optional comma following terminating statement number, optional terminating statement number, etc.)

- Assumed-size array declarators. One can say, for example, "DIMENSION A(1:N,1:*)." This allows a subroutine to be written to assume any size second dimension for A.

- ASSIGN of FORMAT statement numbers: you can use "ASSIGN 10 TO LABEL" in a section of code, and then use, for example, "WRITE(6,LABEL) FOO" in another section of code.

- INTRINSIC statement; FORTRAN-77 semantics for the EXTERNAL statement.
- SAVE statement to preserve values stored in variables, arrays or common blocks after execution of a RETURN or END statement in a subprogram. This is useful when overlays are used.
- Support of null argument lists for functions, that is, a function no longer requires dummy arguments.
- Support of FORTRAN-77 statement ordering rules. See page 6-3 of the DEC-20 FORTRAN manual.
- Compile time constant expressions in declarations, as array bounds and string bounds.
- FORTRAN-77 PARAMETER statements, for defining symbolic constants.
- Expressions as specifier values in OPEN statements and in control information lists in I/O statements.
- Optional link-time typechecking of subprogram arguments.
- New functionality in the ERRSET subroutine.
- Utility subroutine to get a free unit number.
- Additional runtime warnings.
- G-floating double precision numbers.
- TOPS-20 style command interface for the compiler.

Note that when using the EXEC commands at Stanford to compile a FORTRAN program, `/DEBUG:ALL` is the default.

21.10. FORTRAN Utility Programs

Several very useful utility programs exist to help you produce better FORTRAN programs. Each takes a FORTRAN program and transforms it into a neater version. You are strongly encouraged to use these programs to produce your own neat, readable, efficient, and portable FORTRAN programs.

Some of the more popular utilities available include:

RENBR Reformats a FORTRAN program for greater readability. Options exist to modify the statement numbers in FORTRAN programs so that these statement numbers become sequential, to convert DEC-style FORTRAN source programs with initial tabs into standard FORTRAN (or vice versa), to indent the ranges of DO loops and IF statements, and to form cross-referenced listings of FORTRAN programs.

```
@renbr
*myprog.for=myprog.for/i:4/t
```

PFORT Verifies that a FORTRAN program complies with standard FORTRAN-66, and indicates areas of potential incompatibility in moving the program to another computer.

FORSTA FORTRAN timing and statement counting system. The FORSTA program itself creates a new version of your FORTRAN source program with statement counting instructions inserted. When executed, this new version of the program produces a listing of the number of times each statement was executed and the amount of CPU time used by each subroutine and function. For details on running FORSTA, give the command `HELP FORSTA`.

22. Using LISP

LISP stands for LISt Processing language. It is a computer programming language possessing many capabilities lacking in most conventional languages. These special capabilities deal with manipulation of highly structured and symbolic information. LISP has been used to great advantage in such areas as artificial intelligence research, symbolic mathematics systems such as REDUCE and MACSYMA, modeling and simulation, and computer language translators.

Two common dialects of LISP are discussed in this section, MACLISP and Common Lisp.

22.1. Documentation

- *The Revised Maclisp Manual*, Kent M. Pitman, May, 1983. This is the newest and most complete reference manual for MACLISP. It is organized very clearly and has information helpful to both novices and experts. Available through the MIT Laboratory for Computer Science publications office as MIT-LCS TR-295.

- *Common Lisp, the Language*, Guy L. Steele Jr. (Digital Press 1984)

- *Lisp*, Patrick Henry Winston and Berthold Klaus Paul Horn (Reading, MA: Addison- Wesley, 1981 and 1984). An introduction to LISP programming and applications. The second edition uses Common Lisp.

- *Lisp 1.5 Primer*, Clark Weissman (Dickenson Publishing Co., 1967). A standard introduction to LISP in its simplest form. Many good examples, although quite outdated.

- *The Anatomy of Lisp*, John Allen (McGraw-Hill Book Co., 1978). An excellent book dealing with theoretical concepts and practical applications of the language. Extensive bibliography.

22.2. MACLISP

MACLISP is a dialect of LISP from MIT's Laboratory for Computer Science. MACLISP is normally run interpreted, but a compiler is available for those who require faster execution time. A good interactive debugger and program stepper both exist. Also there is an interface to EMACS available through the function `LEDIT`; documentation on how to use LEDIT can be found in the INFO subsystem of EMACS.

MACLISP provides extensive macro and object oriented structure defining facilities not available in all LISP dialects. Novice users need not worry about their existence; more experienced users are referred to *The Revised Maclisp Manual* for further information.

The directory PS:<MACLISP> contains all the source files for MACLISP. Experienced users who find that MACLISP seems to be missing features that they are expecting (DEFSTRUCT, for example) may find the sources on that directory. They can be loaded into the LISP environment via the function `LOAD`.

22.2.1. Running MACLISP

You can run MACLISP by typing `MACLISP` to the "`@`" prompt. LISP assumes that different users will have different requirements in terms of the storage they need and other useful features. Hence, when LISP is started up, it will ask the user for allocation requirements. To use the defaults, simply type "N" to the question "Alloc?" and LISP will take care of setting up an initial environment for you. After the initial "*" is printed, no prompt is used by MACLISP.

There is one very useful change you will probably want to make to your LISP environment. Normally the read-eval-print loop is invoked as soon as the last ")" is entered for an S-expression, that is, no carriage return is required. An unfortunate side effect of this feature is a lack of decent rubout handling. At the expense of having to type a carriage return to wake up LISP, you can get reasonable rubout handling by typing the magic incantation: "`(SSTATUS LINMODE T)`"

As with all LISP interpreters, the top level of MACLISP is a read-eval-print loop. Type in an S-expression and LISP will

evaluate it and return the result to the terminal (unless the user specifies that the output is to go elsewhere). To exit MACLISP, type CTRL/Z, CTRL/C, or (quit).

Sometimes you may encounter an error and a breakpoint will occur. The system will type a message preceded by a semi-colon (";"). There are different alternatives depending on the urgency of the error encountered; an explanation of such options is beyond the scope of this manual. Usually the information that is typed will alert you to the cause of the error. If not, you should read about errors and debugging aids in the manual. In any case, to get out of an error break and return to top level, type a CTRL/G and the system will print out "QUIT*". If for some reason, one CTRL/G isn't sufficient, try typing it a couple of times.

22.2.2. Loading Files and the MACLISP Compiler

It is possible to create a file of LISP functions using your favorite editor and then load that file into your LISP environment. The function LOAD is provided for this purpose. Interpreted LISP files normally have the default extension of LSP. Compiled files have the FASL. If you have a file named FOO.LSP that contains LISP functions, typing "(LOAD "FOO")" or "(LOAD "FOO.LSP")" while in LISP will cause the file to be read into your environment. LISP always prefers FASL files to LSP files, so if you have both the file FOO.FASL and the file FOO.LSP in your directory, and type "(LOAD "FOO")", FOO.FASL will be loaded. You must type "(LOAD "FOO.LSP")" if that is the file you desire. Of course, you are allowed to use whatever names you want for your files, but if you need the file FOO.BAR loaded, the extension must be explicitly specified.

A word of caution: do not forget to enclose file names containing a dot in double quotes. The dot character has a very special meaning to LISP. If found in the wrong place, the cryptic error message "DOT CONTEXT ERROR" will be given.

To invoke the MACLISP compiler, type "COMPLR" to the "@" prompt. When greeted with the underbar prompt, enter the name of the file to be compiled. The default extension is LSP. The compiler will write a new file with the same first name as the one given and the extension FASL containing the compiled code. It may also create a file with the extension UNFASL that contains a record of any problems it may have encountered in the compilation.

22.2.3. A Sample Session with MACLISP

In the example below, what the user typed is underlined. Parenthetical remarks are preceded by a semicolon and are italicized. The rest is typed out by MACLISP.

```
@maclisp

LISP 2122
Alloc? n

*
(sstatus linmode t)T

;type in some simple things and see what happens...
(quote (this is a list))
(THIS IS A LIST)

'(this is a list too)
(THIS IS A LIST TOO)

(cons 'a 'b)
(A . B)

;load in a file of silly functions
(load 'foo)
T
```

```
;let's see a couple of functions we've defined
(grindef firstatom)

;Loading GRINDEF 462

(DEFUN FIRSTATOM (X) (COND ((ATOM X) X) (T (FIRSTATOM (CAR X)))))*


(grindef firstnonatom)


(DEFUN FIRSTNONATOM (X)
  (COND ((ATOM X) NIL) ((ATOM (CAR X)) (FIRSTATOM (CDR X))) (T (CAR X))))*

(firstatom '(a b))
A

(firstnonatom '(a b (c d) (e f) g))
B
;oops! there's a bug in FIRSTNONATOM
;let's call an editor to fix the file.

(ledit)

;loading LEDIT 43
;loading SUBFORK 56
;loading DEFVSY 83
;loading EXTSTR 91
;loading DEFMAX 98
[LEDIT Created.]

....here we used EMACS to modify the incorrect function and
read the new definition into our environment...

[LEDIT Completed.]*

;now let's try it...
(firstnonatom '(a b (c d e) f)))
(C D E)
;better!

;example of simple tracing of a function...
(trace firstatom)

;loading TRACE 67(FIRSTATOM)
(firstatom '(((((a b c d)))))))

(1 ENTER FIRSTATOM ((((((A B C D))))))
  (2 ENTER FIRSTATOM (((((A B C D)))))
    (3 ENTER FIRSTATOM ((((A B C D))))
      (4 ENTER FIRSTATOM (((A B C D))))
        (5 ENTER FIRSTATOM ((A B C D)))
          (6 ENTER FIRSTATOM (A))
          (6 EXIT FIRSTATOM A)
        (5 EXIT FIRSTATOM A)
      (4 EXIT FIRSTATOM A)
    (3 EXIT FIRSTATOM A)
  (2 EXIT FIRSTATOM A)
(1 EXIT FIRSTATOM A) A

;examples of a couple of common errors...

unbound-variable-typed-in

;UNBOUND-VARIABLE-TYPED-IN UNBOUND VARIABLE

;BKPT UNBND-VRBL
QUIT*
```

```
(undefined-function-called-on silly-argument)
;UNDEFINED-FUNCTION-CALLED-ON UNDEFINED FUNCTION OBJECT
;BKPT UNDF-FNCTN
QUIT*
;Now we type CTRL/Z to get out of MACLISP
^Z
@
```

22.3. CLISP

CLISP, the DEC-20 implementation of Common Lisp, is a successor to MACLISP. This version of Common Lisp comes from Charles Hedrick at Rutgers University. Detailed documentation on CLISP's implementation can be found in the file DOC:CLISP.DOC. CLISP conforms to the Common Lisp standard, with the exception that, as of this writing, the irrational math functions such as sine and cosine have not been implemented. There is no compiler.

22.3.1. Running CLISP

You enter CLISP by typing `CLISP` to the EXEC. To exit, use `CTRL/C`. CLISP has a simple EVAL top level which, unlike MACLISP's, includes a prompt. You type Lisp forms to it, and it prints the result. If the form returns multiple values, you will see all of them, each on a new line.

Typing ? should usually give you useful information about the context you are currently in. In many cases you will have to type a carriage return after the ?. At the top level, it tells you how to define functions, and describes some of the most important facilities. In other situations ? is rebound to messages that are useful in that context.

You can enter EMACS from CLISP to edit your functions. Typing `(ed 'function-name)` will put you into an EMACS buffer containing that function. When you exit EMACS that function is automatically reloaded. Typing just `(ED)` will put you into an empty EMACS buffer or into your previous one. Typing `(ED 'filename)` will put you into EMACS with that file in the buffer. You must save any changes before you exit back to CLISP and then re-LOAD the file.

22.3.2. The CLISP Error Handler

When you make an error, you get an error message and are put into a sub-level where you are given a chance to redeem yourself. The following special commands are available in this level.

?	Print this list.
BK	Show the active calls (the stack).
^^	Return to top level.
OK	Retry the failed operation, using NIL.
(RETURN <value>)	Retry the failed operation, using <value>.

Below is an example session using the error handler. Comments are preceded by semi-colons and the user's typing is underlined.

```
;An undefined variable

CL>oops

Error in function OOPS.
Unbound variable: OOPS
If continued: Please define it before continuing
1> (setq oops '(i forgive you))
(I FORGIVE YOU)

;Retry the operation with the new information

1>ok
(I FORGIVE YOU)
CL>(undefined-functions also give us an error)

Error in function UNDEFINED-FUNCTIONS.
Undefined function: UNDEFINED-FUNCTIONS
If continued: Please define it before continuing

;This time let's just abort back to top level

1>^^
CL>
```

22.3.3. Debugging Facilities for CLISP

There are several debugging aids available within CLISP. They are the built-in TRACE and STEP functions. TRACE allows you to ask for a printout whenever a certain function is called. The printout shows the arguments with which it is called and the value returned. This is very useful for trying to keep track of what a certain function (or functions) is doing. (UNTRACE function) stops tracing. Just (UNTRACE) will stop tracing of all functions.

STEP is a single step facility which allows you to watch the interpreter EVAL each form individually.

22.3.4. Sample CLISP session

Following is a sample session with CLISP. Comments follow the semi-colons. What the user types is underlined.

```
@clisp
Common Lisp
Copyright (C) 1984, Charles L. Hedrick

;Unlike Maclisp, CLISP has a prompt. Let's try some simple things.

CL>(quote (this is a list))
(THIS IS A LIST)
CL>(setq test '(this is a test))
(THIS IS A TEST)
CL>test
(THIS IS A TEST)

;We need a function to work with so let's define one...

CL>(defun factorial (n) (cond ((zerop n) 1) (t (* n (factorial (1- n))))))
T

;We could also have loaded a file containing the function definition by saying (load 'foo.lsp)

CL>(factorial 4)
6
```

```
;So we can see it in action

CL>(trace factorial)
FACTORIAL
CL>(factorial 3)

;The number 0: is the level of the call to the function

0: (FACTORIAL 3)

1: (FACTORIAL 2)

2: (FACTORIAL 1)

3: (FACTORIAL 0)

3: returned 1

2: returned 1

1: returned 2

0: returned 6
6

;To keep our output simple, disable the tracing output

CL>(untrace factorial)
FACTORIAL

;Now go through the execution step by step.  More useful when you have a function that doesn't work.

CL>(step (factorial 3))
(FACTORIAL 3) : n
   3 = 3
   (BLOCK FACTORIAL (COND # #)) : n
    (COND ((ZEROP N) 1) #) : n
     (ZEROP N) : n
   N = 3
   NIL
   T = T
   (* N (FACTORIAL #)) : n
   N = 3
   (FACTORIAL (1- N)) : s
   2
    6
   6
  6
 6
6

;When you are done, type CTRL/C to exit.
CL>
^C
@
```

23. Using the Mail System

You can use the computer to send messages to other users which will be stored for them to read at their leisure. Each user has a "mailbox", the file MAIL.TXT.1 on his or her directory. You can use the MAIL command to send messages to other user's mailboxes, and you can use the TYPE command, or a program such as MM, to look at the messages in your mailbox.

23.1. Sending Mail

The MAIL command allows users to send messages to other users (whether logged in or not) to be read at their convenience[33]. To send a message to another user on the same computer as you are logged onto give the MAIL command followed by the user name. If you don't already know, you can find out a person's user name from his or her real name by giving the command FINGER followed by a space and their last name.

The MAIL command normally prompts with two questions:

Subject: A single line summary of the message to be sent.

Msg: The body of the message to be sent. This is a collection of text normally terminated by CTRL/Z.

In the following example, a user on the LOTSA computer sends mail to P.POTEET reporting a system problem:

```
@mail p.poteet
 Subject: LOTS mess
 Message (end with FSCAPE to get to MM command level, CTRL/Z to send):
The mess in the CERAS printer room is awful.  How about someone cleaning
up the paper around the line printer?
^Z
p.poteet@LOTS-A.#Pup -- queued
@
```

To send mail to a user that is on another computer, you would indicate which computer he or she is on by appending an "@" and the name of that computer to the user name. Thus, if F.FRANK (who has his account on LOTSA) wants to send mail to J.JOE-COOL, whose account is on LOTSB, Frank would do as follows:

```
@mail j.joe-cool@lotsb
```

If you just give the command MAIL with no user name, the system will prompt for To: and Cc: (carbon-copy) lists before asking for a subject. These lists consist of one or more user names separated by commas. To the To:, Cc:, and Subject: prompts, you may also respond with what is called an *indirect file*, a file containing a list of recipients. The file is specified by typing an at-sign (@) followed by the file name and a space. You can use several indirect file inclusions per message. In the example below, CLASS.DIS is the indirect file and contains a list of user names, arranged just as they would be had the user sending the mail typed them in directly.

```
@mail
 To: @class.dis
 Cc: b.bander@lotsb, j.irandom@lotsa
 Subject: CS105-1 due dates
```

While typing the body of a message, several special characters may be typed in addition to the usual editing characters (CTRL/U, CTRL/W, CTRL/R, and DELETE):

- To display the whole text of the message, type CTRL/L. CTRL/R, as usual, displays just the last line.
- To insert text from a file into the body of a message, type CTRL/B. You will be prompted for a file name to insert.
- You may edit the text of your message by typing CTRL/E. When you exit from the editor (via E or EU from

[33] The TALK command provides communication of extended duration with logged in users; try the EXEC command HELP TALK.

EDIT) you will be returned to the mail program.

- If you end your message with ESC instead of CTRL/Z, you will be prompted with an S>, and given a chance to enter commands to modify the subject, text, and list of recipients. To the S> prompt, try typing HELP for more information. To send the message, just press RETURN. To abort the message, type QUIT.

23.2. Reading your Mail

When you log in, you will be notified if you have received mail in your mailbox since the last time you read your mail. Various mail reading programs are available to allow you to look at your mail, delete it, reply to it, and so on.

Your mail is stored in a file called MAIL.TXT.1, which is for many purposes an ordinary file that you can DELETE, PRINT on the line printer, or TYPE on your terminal. DELETE works a bit differently with mail files, though: when you delete your mail file, its contents are automatically expunged, so undeleting does not work.

Warning:

You must NOT try to edit your mailbox file. Although mail files look like ordinary text, they actually have a special internal format, and editing them is almost guaranteed to make them unreadable, as well as to make it impossible for you to receive mail.

The fastest way to read your mail is to give the command

```
@type mail.txt
```

After you have read your mail, you may want to delete it with the command

```
@delete mail.txt
```

Most people use MM, a powerful mail-handling program, to read their mail. MM can be used to read your mail message by message, reply to selected messages, and delete selected messages. MM has an extensive set of internal HELP commands, which you are encouraged to use to learn more about MM. The MM manual may be found in the file DOC:MM.DOC.

In normal usage, MM gives a "MM>" prompt when it is waiting for commands which affect the mail reading session as a whole. Among the more useful such commands are "READ" to read a list of messages, "EXIT" to leave MM, and "HELP". When you use the READ command, MM will display each message in succession, and after each message will pause with a "R>" prompt to allow you to deal with that particular message. To the "R>" prompt you may give such commands as DELETE (to remove that message from your mailbox), REPLY (to set up a response), FORWARD (to pass the message on to somone else), or you may just press RETURN to move onto the next message in the list.

The following illustrates a typical use of a few of MM's features. P.POTEET reads his mail, and sends a reply to a message. If he had wanted more help, he might have said HELP, followed by a topic (e.g. HELP DELETE, or HELP ? to see the possible help topics), to the MM> or R> prompt.

```
@mm
LOTS-A.#Pup MM Version 5.2(1657), Edit 943
N     71  5 Aug F.FRANK Franklin Bell LOTS mess (257 chars)

 Last read:  5-Aug-84 08:06:27, 71 msgs (70 old), 22 pages
 8 messages deleted
MM>read                                          READ with no arguments means read each new message.
 Message 71 (257 chars):
Date: Sun 5 Aug 84 10:38:27-PDT
From: Franklin Bell <F.FRANK@SU-LOTS-A>
Subject: LOTS mess
To: P.POTEET

I recently sent a gripe complaining about the mess in the CFRAS printer
room, and got no response.  How about someone cleaning up the paper around
the line printer?
-------
R>reply                                          He decides to answer this message.
 Message (end with ESCAPE to get to MM command level, CTRL/Z to send):
LOTS stands for LOW OVERHEAD Time Sharing, and that means the staff is
very limited.  The line printers are self service, and users are expected
to clean up their own, and other people's, messes.  Please help!
^Z
 F.FRANK@LOTS-A.#Pup -- queued
R>delete                                         Now that it's been answered, may as well delete it.
R>                                               A carriage return moves to next message.
                                                 There were no more new messages, so we end up
                                                 back at the top level prompt.
MM>exit                                          All done.  Time to quit.
 Expunging deleted messages
@
```

Through the use of the MM.INIT and MM.CMD files, you can tailor MM's prompts, message headers, capitalization, and so forth. Refer to the MM manual for details.

23.3. Sending Messages to Logged-in Users

If you want to send a short message to someone who is currently logged in, you can use the SEND command[34].

```
@send p.poteet Thanks for your reply.  I'll try to help.
```

appears on P.POTEET's terminal as:

```
F.FRANK, TTY26, 8-Aug-84 10:40PM
Thanks for your reply.  I'll try to help.
@
```

If P.POTEET had given the command REFUSE SYSTEM-MESSAGES, then F.FRANK would instead have received an error message indicating that the terminal message had been refused.

The programs WHAT and REPLY are useful for handling terminal messages. All terminal messages are written to a temporary file called SENDS.TXT in the recipient's directory. Using the WHAT program you can peruse the contents of that file. For example, typing "WHAT" to the EXEC will cause the last terminal message you received to be typed on your screen. This is very handy if, for example, you were refreshing your screen in EMACS when the original message was sent. The REPLY program is used to send a terminal message back to the person who sent the last message in your SENDS.TXT file. See the HELP files on SEND, WHAT, and REPLY for more details.

23.4. BBOARD -- The On-line Bulletin Board

Every system has an on-line bulletin board that can be read by all users. The bulletin board contains messages about housing, items for sale, upcoming events, political debates, and other items of interest. To read the bulletins run the BBOARD program by giving the EXEC command BBOARD.

---

[34] The TO command is a synonym for the SEND command

The procedure for posting messages on the bulletin board varies from system to system. At LOTS use the program POSTBB. This will assure that your message gets to all the LOTS machines. Use of this program is self explanatory. On Score, Sierra, SUMEX, and CSLI you should send your message to SU-BBOARDS for redistribution among a dozen or so research computers. It doesn't matter if you think your message applies only to users of your system. Trying to send to BBOARD will result in getting your mail returned with a scolding. The GSB DEC-20's have their own scheme for handling bulletin board messages.

You may use the BBOARD program to read other system mailboxes or special interest bulletin boards. The command BBOARD SYSTEM will display any system messages that may have been sent since you logged in. A DIRECTORY command of the directory PS:<BBOARD> will give you an idea of what special interest bulletin boards are available on your system. If, for example, one of the files was GRIPES.TXT, then the command BBOARD GRIPES would be used to read those messages.

24. Using MIC Command Files

MIC is an acronym for Macro Interpreted Commands. The MIC facility provides a way of executing command files on a terminal. It is more powerful than the TAKE command in that you can include program commands in the MIC file instead of just EXEC commands. You can watch the progress of the command file's execution and can interact with the MIC processing to a degree. It is also possible to give arguments to a MIC file, allowing for great flexibility.

The MIC facility accepts command files with the same format as batch control files. A file that runs under the batch system will run under the MIC facility. MIC is less picky than the batch system about distinguishing between EXEC and user commands; it is possible to leave out the initial "@" and "*" characters that the batch system insists upon. The default file extension for MIC command files is .MIC.

24.1. Running a MIC file

A MIC command file is invoked by the EXEC DO command followed by the name of the command file and an optional list of parameters separated by commas. Below is an example of the EXEC command line that runs a MIC file to save files on a tape drive.

```
@do dump.mic MTA1:,since 21-Aug-84
```

The above example has two parameters, a tape drive name and a DUMPER command. In a command file, a parameter is represented by a single quote followed by a letter. 'A is the first parameter, 'B is the second parameter, and so on. You can use only the letters A through Z. Below is an example of a MIC command file.

```
@assign 'A                    The tape drive name is used here
@dumper
*eot
*'B                           The DUMPER command is used here
*save ps:<f.frank>*.*.*
*rewind
*unload
*exit
@deassign 'A                  The tape drive name is used again
```

It is also possible to have a MIC file without any parameters.

MIC command files can be nested by simply including a DO command within the outer command file. Note that the nested command file will have a completely new set of parameters and will not have direct access to the previous set. This means that any parameters of the outer command file which need to be accessed by the inner macro will have to be passed as parameters in the DO command.

When the MIC file is done running, the message "[MICEMF - End of MIC File: FILE.TYP.GEN]" will be printed, where FILE.TYP.GEN is the filename used in the DO command.

The EXEC's KMIC command cleans up MIC processing. It is usually typed automatically by the EXEC when the MIC file finishes. The KMIC command ensures that another DO command can be given without MIC nesting taking place.

24.2. Interacting with MIC

It is possible to interact with MIC in a limited fashion. A MIC file can be suspended, continued or aborted.

The execution of a MIC command file can be suspended by typing CTRL/B once or twice. Like CTR/C, typing one CTRL/B will take effect when the EXEC or program next requires input; typing two will take effect right away. The message "[MICBRK - MIC is breaking]" indicates that commands are no longer being executed from the file. At this point it is possible to type in commands without worry about interacting with the MIC processor. At other times your characters would get mixed up with commands from the MIC file, usually producing a mess

You can restart a suspended MIC file by typing one or two CTRL/P's. The message "[MICPRC - MIC is proceeding]"

indicates that the MIC file has restarted.

A MIC command file can be aborted by typing CTRL/A once or twice. You will get the message "[MICABT - MIC is aborting]" when the file is aborted. If you have nested DO commands, you will need to either abort each nesting with CTRL/A or use the KMIC command to shutdown all MIC processing. When a MIC command file terminates normally, i.e., it comes to the end of the command file, the EXEC executes a KMIC command to ensure that the next DO command will not be affected by the previous DO command.

25. MLAB - A Modeling Laboratory

MLAB is a general purpose modeling laboratory that allows a user to make up functions, fit data to those functions, and plot the results on a graphics terminal or lineprinter. At present MLAB is available only at LOTS.

These few pages are by no means meant to be a complete guide to MLAB. The user who wishes more information can type HELP to the MLAB prompt (the MLAB prompt is a "*"). MLAB will then direct the user to the appropriate portion of its help file.

MLAB may be invoked by typing MLAB to the EXEC prompt. MLAB is exited by typing EXIT to the MLAB prompt.

If you are interested in using the graphics capabilities of MLAB, it is best to use it on a DEC GIGI terminal when at LOTS. MLAB also works well on Tektronix display terminals. These terminals allow you to view plots *exactly* as the plots will appear printed. Other terminals may only allow a rough sketch of the plot to be seen.

25.1. Expressions and Statements

An example of an expression is:

```
A+B
```

An example of a statement is:

```
X=A+B
```

It is important to note the distinction between an expression and a statement. One uses MLAB by typing statements. A statement actually *creates* the variable on the left side, and *assigns* it the value of the expression on the right side.

```
A=4
B=7
X=A+B
```

Expressions are used as the right sides of statements, or as the argument to the TYPE command. The TYPE command may be used in order to make MLAB act as a calculator, for example:

```
T=4
TYPE 4*T+2
```

MLAB would then type back "18", in accordance with the normal operator precedence for addition and multiplication. If one is unsure of the order in which an expression will be evaluated, one should use parentheses. Beware that typing an expression by itself to MLAB is pure nonsense.

25.2. Variables

MLAB is much like BASIC in that variables and their "types" do not have to be "declared". Simply assigning a variable a value causes MLAB to figure out whether the variable is a scalar, matrix, and so forth. For example,

```
PI=3.14159
```

will create a scalar variable called PI and assign it the value 3.14159. To change a defined variable's value, simply reenter the assignment:

```
PI=4
```

The value of any variable may be seen using the TYPE command:

```
TYPE PI
```

MLAB would type back the value "4", in this case.

If a variable is no longer needed, it may be deleted by using the DELETE command. To delete PI, for example, one would type:

```
DELETE PI
```

25.3. Reading Data

Use the READ command to read data from a file or the terminal into a variable. The form of the READ command is:

VARIABLE=READ("FILENAME.EXT",ROWS,COLS)

This will read data from the file, FILENAME.EXT, into the variable, VARIABLE. MLAB will continue reading until it runs out of data, or ROWS*COLS numbers have been read. COLS should always be the *exact* number of columns or disaster may result. If COLS is not specified, COLS is assumed to be one. If ROWS is not specified, ROWS is assumed to be two thousand. Keep in mind that MLAB can only handle a finite number of points.

As an example, if a file called RESULTS.OUT had data that looked like

```
4 10 16
5 8  62
6 38 4
7 81 33
```

then RESULTS.OUT could be read into a variable named MATRX by typing:

```
MATRX=READ("RESULTS.OUT",4,3)
```

If the file name was TTY:, then MLAB would read from the terminal until a CTRL/Z was typed. In order to read those same numbers from the terminal, instead of a file, you would type:

```
MATRX=READ("TTY:",4,3)
4   10   16
5   8    62
6   38   4
7   81   33
↑Z
```

When dealing with matrices, the COL or ROW built-in functions, along with the START:FINISH:INCREMENT built-in operator are very useful.

```
M COL 1=1 : 2.75 : 0.5
M COL 2=2 : 4
```

This would create a two column matrix called M which would look like:

```
1.0 2.0
1.5 3.0
2.0 4.0
2.5 0.0
```

Notice the effect of the increment of 0.5 in column one. In column two there was no increment specified, so the default increment of 1.0 was used. Notice also that column one did not "run" past the finish value. In column two MLAB automatically set M(4,2)=0.0 in order to have a full matrix.

To find out more about variables, type **HELP** *to the MLAB prompt. Then type* **GOTO VARIABLES**. *See the next section on where to find more information on the various pre-defined functions used to read data from a file or terminal.*

25.4. Built-in Operators and Predefined Functions

The user of MLAB usually wants to have MLAB perform calculations on data. MLAB has a set of built-in operators and functions to aid in the process. A few examples of operators and functions have already been seen: "+" and "*" are operators, while READ is a predefined function. This section is intended for use as a reference. For more information, see the help file noted at the end of this section.

In the following table, A and B are either scalars or matrices, C is a scalar and S1, S2, and S3 are statements. All examples here use only expressions; statements would be required in actual use.

The following operators may be used for both scalars and matrices:

A + B	A < B	A AND B
A - B	A > B	A OR B
- A	A = B	IF S1 THEN S2 ELSE S3
	A <= B	A:B
A / B	A >= B	A:B:C
A ↑ B	A NOT= B	A * B
ABS A	NOT A	A ↑↑ B
A ROW B	A COL B	

In the following table, A and B are matrices, I and J are scalars and F is a function.

A(I,J)	Specifies an element of A.
A &' B	Column concatenation of A before B, that is, if A=1,2 and B=3,4, then A&'B is 1,2,3,4.
A & B	Row concatenations of A over B.
A # B	Returns matrix whose I-th row is the vector cross product of A's and B's corresponding rows.
A'	Transpose of A.
A *' B	Multiply matrix A by a scalar which depends on that row.
F ON A	Returns result of the function F applied successively to each row of matrix A.

Here is a list of most MLAB functions, along with their descriptions. X and Y are scalars. For more information on the use of these functions, see the help file listed at the end of this section. The functions are presented in expression form.

SIN(X)	Sine of X, where X is in radians.
SIND(X)	Sine of X, where X is in degrees.
COS(X)	Cosine of X, where X is in radians.
COSD(X)	Cosine of X, where X is in degrees.
ATAN(X)	Arc tangent of X (in the boundary $-\pi/2$ to $\pi/2$).
ATAN2(X,Y)	Arc tangent of Y/X. Complete result in the range $-\pi$ to π. ATAN2(X,Y) is the angle formed by the vector to the point (X,Y) and the X axis.
ACOS(X)	Arc cosine of X.
SINH(X)	Hyperbolic sine of X.
COSH(X)	Hyperbolic cosine of X.
TANH(X)	Hyperbolic tangent of X.
SQRT(X)	Square root of X.

LOG(X) Natural log of X.

LOG10(X) Common log of X (base 10).

EXP(X) *e* to the power X.

INT(X) The greatest integer less than or equal to X.

MOD(X,Y) INT(X)-INT(INT(X)/INT(Y))*INT(Y).

RAN(X) Generates pseudo-random numbers.

NROWS(M) Returns number of rows in the matrix M.

NCOLS(M) Returns number of columns in matrix M.

LIST(R1,R2,...) Generates a one column matrix that is concatenation of the scalar or matrix arguments..

DIAG(M) Generates a one column matrix containing the longest diagonal of M.

LOGGAMMA(X) Result is the log of the gamma function. X must be positive.

POINTS(F,M) Used in obtaining matrices for drawing curves from functions. See page 168 for an example.

COMPRESS(M) Generates a copy of the matrix M without the rows whose first element is zero.

TYPEOUT(X) Y=TYPEOUT(X) sets Y=X and types X on the terminal.

READ(FILE.EXT,R,C)
Reads data from a terminal or file. See page 164 for an example.

SVD(M) Singular value decomposition of matrix M. See the help files for more information.

INTEGRAL(X,A,B,E)
Definite integral of expression E with respect to X on [A,B]. Can only be used in function bodies of user defined functions.

INTERPOLATE(M,N)
Returns a matrix as if a smooth curve was passsed through M. N will be the first column.

CROSS(M,N) Each row of M is column-concatenated with each row of N. Resulting vectors form rows of CROSS(M,N).

SUM(I,A,B,E) Generates sum of expression E with I running from A to B.[2]

HISTO(M) Returns a plottable curve matrix for the histogram of M, which must be sorted by column one.

MESH(M,N) Alternate rows of M and N returned in a matrix. Unfilled elements set to zero.

ROOT(X,A,B,E) Returns value V which makes expression E(X)=0 if X=V. A<=V<=B.

CURVEMATRIX(C)
Returns matrix defining curve, C. See section 25.7

ROTATE(M,X) Returns the rotated matrix M ROW ((NROWS(M)+1-X):NROWS(M),1:NROWS(M)-X.

EVAL(X,A,E) Substitutes expression A for all occurrances of X in expression E.

LABELMATRIX(C)
Returns a copy of the matrix containing the curve C's label. See page 168

COV(M) Returns covariance of NCOLS(M) observations using M.

LOOKUP(A,M) Returns the Y value if a smooth curve was passed through M, and A was the X value.

CORR(M) Returns correlation matrix of the observations which are the rows of M.

CONTOUR(M,L) Returns the 2D contour of a 3D plot. L contains steps.

SORT(M,X) Sorts M so that the elements of column X are in increasing order.

DATATYPE(E) Returns the type of object E, e.g., scalar, matrix, and so forth. Not for use in functions.

SMOOTH(M) Returns a "smoothed" version of M. End points may be untrustworthy.

INTEGRATE(F1 DIFF X, F2 DIFF X, ..., FN DIFF X,M)
First order integrals. See the help files.

To find out more about built-in operators and functions, type **HELP** *to the MLAB prompt. Then type* **GOTO BUILT-INS**

25.5. Creating Functions

Creating functions is easy using the FUNCTION command along with MLAB's operators and functions. MLAB functions can only return scalar values, not matrices. The MLAB user who is serious about creating functions should read the help files to become thoroughly familiar with MLAB's built-in operators and functions. Below is an example using a function.

```
FUNCTION F(X,Y)=X↑2+X/2+A
A=3
TEMP=F(2)
TYPE TEMP
```

MLAB would type back "8". Notice that the constant A did not need to be assigned a value before the function declaration, but it *did* need to be assigned before the function was *used* in line 3 in the example above. Notice also that the parameter Y was not used in the function, although it appeared in the parameter list. When declaring a function, a parameter list *must* be supplied, even if it is empty, or none of the parameters are actually used.

In the following example

```
A=1
FUNCTION F()=23+A
FORCE=F()
TYPE FORCE
```

MLAB would type back "24". Again A was assigned a value before the function was used, but it also happened to be assigned before the declaration of the function as well.

Now, if

```
TYPE F
```

is typed, MLAB will type back "23+A".

Functions may be passed as parameters to other functions, or may be passed recursively, as shown below.

```
FUNCTION FACTORIAL(X)=IF X=0 THEN 1 ELSE X*FACTORIAL(X-1)
```

To find out more about creating functions, type **HELP** *to the MLAB prompt, then type* **GOTO CREATING-FUNCTIONS**

25.6. Curve Fitting

Using the FIT command you can force MLAB to adjust the constants in any function in order to fit a set of data.

The data must be in a matrix in the same order as the parameters of the function with the dependent variable last. In other words if the function looks something like the following

```
FUNCTION F(X,Y,Z)=A*X+B*Y+C*Z
```

then the matrix must look something like

```
X1  Y1  Z1  F(X1,Y1,Z1)
X2  Y2  Z2  F(X2,Y2,Z2)
X3  Y3  Z3  F(X3,Y3,Z3)
...
```

Let us call the above matrix M and assume that the data is already in M. Remember that a function cannot be used until the constants exist, so they must be assigned initial guesses. Since the fitting algorithm is quite good, your guesses don't have to be accurate at all, but the constants (A, B, C in the above example) *must* exist.

```
A=1
B=3
C=10
```

To fit the function to the data, type:

```
FIT(A,B,C),F TO M
```

If A and C only require adjustment, perhaps because B should be held at its current value of 3, you would type:

```
FIT(A,C),F TO M
```

MLAB will ask three questions. For most purposes the default answers are adequate, so just press the RETURN in response to the questions. The defaults are displayed in parentheses. MLAB will then type out some intermediate results and after a while type the values of the constants it was asked to fit. MLAB will actually change the values of those constants to their fitted values, losing the original guesses.

If you do not want to see the intermediate results, type QUIET or SILENT before the FIT command.

```
QUIET FIT(A,C),F TO M
```

To find out more about curve fitting, type **HELP** *to the MLAB prompt, then type* **GOTO CURVE-FITTING.**

25.7. Plots on a Terminal

Once a matrix containing the points to be plotted exists, the matrix may be plotted using the DRAW command. For example to draw the two or three dimensional matrix L into the plot, type:

```
DRAW L
```

If a multi-column matrix called W existed, and you wanted to plot the eighth column as the Y-values, you would type:

```
DRAW W COL 8
```

The X-values are always assumed to be in the first column.

To draw any single-parameter function, use the START:FINISH:INCREMENT operator, along with POINTS (a built-in function) as follows:

```
DRAW POINTS(SIN,1:10:0.1)
```

This command would draw the function, SIN, from 1 to 10 in increments of 0.1. Of course a user-defined function may be used with POINTS in the same fashion, but it must be a single-parameter function. If you want to use a multi-parameter function, the function first must be used to make a matrix.

The DRAW command creates a *curve* inside a *window*. Curves and windows are variables, just like scalars and matrices. A curve is a set of plottable points and a window is a set of curves. There are two default windows: DEFAULTWINDOW and DEFAULT3DWINDOW. The latter is used for all 3-D curves, the former for all 2-D curves. All curves are automatically added to the window. Thus if DRAW is used twice, two curves are drawn in the window. MLAB calls most curves things like TEMPC1, TEMPC2, etc. Thus, to delete the second curve drawn, you would type:

```
DELETE TEMPC2
```

To see the names of the curves in the window (which is, in this example, DEFAULTWINDOW), type:

```
TYPE DEFAULTWINDOW
```

To temporarily blank the second curve from the window, so that it could be retrieved later, type:

```
BLANK TEMPC2
```

To retrieve a suppressed curve, you would type:

```
UNBLANK TEMPC2
```

If a GIGI terminal is being used, and you want to see both windows (which contain all the curves) drawn on the screen, simply type:

```
DISPLAY=GIGI
```

Of course this means that the DRAW command must have been used first or else nothing would be displayed.

Warning:

While MLAB is sending the plot to the screen, the terminal is set in a peculiar state. Do not type CTRL/C. ***If*** CTRL/C ***is accidentally typed, type*** CONTINUE ***or the terminal may not do anything at all!***

Interrupting messages from all sources may cause the terminal to draw strange lines, so it may be desirable to disable interrupting messages with the EXEC command REFUSE SYSTEM-MESSAGES ***command before using the MLAB*** DRAW ***command.***

If you are not using a graphics terminal, use the TTYDRAW command instead. Beware, however, that the DRAW command should be used in order to obtain a hardcopy plot, because the TTYDRAW command does not add a curve to the window.

25.8. Labeling a Plot

To label the plot after using the DRAW command, the STRING command may be used. The STRING command places a text string in the most recent curve, and thus, the window. The point where the center of the leftmost character will appear is specified. A point outside the boundaries of the graph, or even outside the visible range of the plot, may be specified. For example:

```
STRING "TEMPERATURE VS TIME" AT -1,4
```

To print the string vertically, one would use the command:

```
VERTICAL STRING "TEMPERATURE VS TIME" AT -1,4
```

There is much more to the DRAW and STRING commands that the serious MLAB user will wish to know. Among the many things DRAW can do is change the point representations and types of lines drawn. The DRAW command can do just about anything needed, whether for a 3-D or a 2-D plot. See the help files for more information.

To find out more about plotting, type **HELP** *to the MLAB prompt. Then type* **GOTO PLOTTING.**

25.9. Obtaining Lineprinter Plots

Once the DRAW command has been used to make a plot, the window may be sent to the printer by typing:

```
DISPLAY=PRINTRONIX
```

Changing the display to type PRINTRONIX is by far the easiest way of obtaining hardcopy plots. Once the display is set to PRINTRONIX, however, all DRAW commands will immediately send the windows to the printer until the display is set back to what it was before.

The other, more elaborate, way of obtaining hardcopy is with the PLOT command. To do so, simply type:

```
PLOT
```

MLAB will make a file called something like 000001.PLX containing the windows. Before anything can be sent to the printer, however, MLAB must be exited, and the programs PLOTX or OMPLOT must be run. PLOTX, described on page 197, allows reformatting of the plot. It then may be used to generate a file called PLOTX.OUT which may then be printed on the Printronix printer. OMPLOT allows the user to view the plot on a graphics terminal or to send it to the Printronix printer. OMPLOT is described in detail on page 198.

To find out more about obtaining lineprinter plots, type **HELP** *to the MLAB prompt. Then type* **GOTO HARDCOPY.**

25.10. Miscellania

MLAB contains a method for saving work between sessions with the SAVE and USE commands. To save the current windows, type:

```
SAVE IN MYLOG
```

MLAB will then save the windows into a file called MYLOG.DAT.

When MLAB is invoked in a later session, you can restore the MLAB variables, functions, and so forth, with the command:

```
USE MYLOG
```

Variables may also be saved. The contents of a variable called LENGTH could be saved by the command:

```
SAVE LENGTH IN MYLOG
```

The authors of MLAB have also assembled a set of error message explanations, as well as some user-tips. These are included in the miscellania help file.

To see the miscellania help file, type **HELP** *to the MLAB prompt. Then type* **GOTO MISCELLANIA.**

26. Using Networks

26.1. Some Terminology and Concepts

A communications system that connects distinct computer systems is called a *network*. A network allows a user to gain access to files and other resources regardless of the user's location. You don't have to be at a local terminal to log onto a computer on a network; you can use an appropriate terminal on the other side of campus or on the other side of the country. Files are easily moved from computer to computer without the time consuming hassles of reading and writing magnetic tapes. Electronic mail can be sent to people using other computers on the network.

The local network that interconnects the Stanford computers is the Stanford Ethernet, or SU-Net. There is a nation wide research network called the Arpanet that connects to some of research computers at Stanford. The Score, Sierra and SUMEX DEC-20's are on both the Arpanet and the SU-Net.

A computer on a network is often referred to as a *host*. If you are using a network program to communicate with another computer, the other computer is referred to as the *remote* or *foreign host*. The computer on which you are running that network program is called the *local host*. On the SU-Net there is a special type of small computer called an *Ethertip* that does nothing but connect terminals to hosts and then supervise the exchange of characters. Ethertips are very useful since they can be used to log onto to any computer on the SU-Net.

All networks use the transmission of electronic signals to achieve their communication function. Sometimes coaxial cable is used (as with the Ethernet), sometimes special purpose phone lines (as with the Arpanet), and even radio signals bounced off of satellites. The manner in which those signals are interpreted is often called a *protocol*. It is very common in networks to build higher protocols out of more primitive protocols. A collection of related protocols is called a *suite* of protocols. For example there may be hardware protocol for interpreting electronic signals as characters. Building upon the hardware protocol, a more advanced character stream protocol could be used to ensure that the characters are delivered to their destination in an orderly fashion. Yet another protocol could be used so that a file (a stream of bytes with a name and other properties) could be moved over the network.

There are two protocol suites in use on the Stanford Ethernet. The first is called PUP (for PARC Universal Packets). At present nearly every host on the Stanford Ethernet can communicate using PUP protocols[35] . The second protocol suite is refered to as TCP/IP (for Transmission Control Protocol/Internet Protocol) or the Internet protocols. The Internet protocols are also used on the Arpanet.

Most network programs are rather smart about figuring out which protocol suite to use. Normally only systems programmers have to worry about which protocol a program is using.

26.2. Network Access

Depending on your reason for using a particular DEC-20, your access to the various networks may be restricted. Controlling access to the Arpanet is the major reason behind these restrictions.

The Arpanet is maintained by the Defense Advanced Research Projects Administration (DARPA) for use by researchers with DARPA or DARPA approved federal funding. DARPA insists that people not associated with DARPA funded projects not be allowed access to the Arpanet and that people with Arpanet access use that network only for DARPA related activities. Failure to observe these access restrictions would result in the loss of Arpanet connections to Stanford. This would be a great hardship to a large number of researchers, hence the access controls.

The details of granting network access vary from system to system. Consult your system staff if you believe you should have network access privileges that you don't already have.

---

[35] As of this writing the only exception is the STAR-VAX in the EE Department. You must use Internet protocols when communicating with that host.

26.3. The TELNET Program

TELNET is a program to allow users to log onto other hosts over a network such as the Stanford Ethernet.

26.3.1. Using TELNET to Log onto Other Computers

The simplest way to run TELNET is to type TELNET followed by the host name to which you wish to connect. Below is an example of using TELNET on LOTSA to log onto the Sierra DEC-20.

```
@telnet sierra
 Trying... Open

 Stanford Sierra, TOPS-20 Monitor 5.3(5715)-4
@lougheed
 Job 24 on TTY145 29-Aug-84  6:47PM
 Previous LOGIN: 29-Aug-84  6:23PM
@
```

If the Sierra computer had not been up, TELNET would have eventually printed an error message to that effect. TELNET tries very hard to make the connection and will wait for over a minute before giving up.

At this point everything you type goes to the computer to which you have connected. Remember that you are still logged onto the computer where you ran TELNET. You should be aware that you may need to set your terminal type at this point using the TERMINAL command (see page 51). When you log off of the computer you connected to, TELNET will print a message indicating that the connection has been closed.

```
@logout
Killed Job 24, User LOUGHEED, TTY 145, at 29-Aug-84 18:47:38
 Used 0:00:01 in 0:00:07

Connection closed by foreign host
@
```

TELNET tries to choose the best path to another system. It is possible, though unlikely, that you would want to override TELNET's choice. Consult the TELNET documentation for examples of how to do this.

26.3.2. TELNET Commands

It is possible to give commands to the TELNET program while you are connected to another host. The simplest way to do this is to type CTRL/^ followed by a command letter[36]. The most useful command characters include:

C	Close the connection and get out of TELNET.
CTRL/^	Send a CTRL/^ character to the foreign host.
P	Push to an EXEC so you can give EXEC commands on your local host.
T	Toggle transparent mode so that EDIT and META keys work properly.
S	Show the status of your connection.
X	Enter extended command mode.

If you are using TELNET from an Ethertip terminal, you will have to type CTRL/^ twice, since the Ethertip also uses CTRL/^ as a command character. For more information on TELNET, consult the file DOC:TELNET.DOC.

---

[36] The control uparrow character is represented on some terminals (notably the VT100) by the CTRL/~ character

26.4. Network Mail

Using the mailsystem on the DEC-20 to send mail to someone on a different host is not terribly different than sending mail to a local user. If you don't know how to send local mail, you may first want to refer to Chapter 23, page 157.

To indicate that the username belongs to a person on a different computer you add an "@" and a *host name* after the username in the "To:" or "Cc:" field. Every computer that you can send mail to has a name; if you get the host name wrong, the mailsystem will tell you right away. Below is an example of sending mail to someone on a foreign host.

```
@mail
 To: lougheed@sierra
 cc:                                <-- Press RETURN if you don't want to send a carbon copy
 Subject: This is a test
 Message (end with ESCAPE to get to MM command level, CTRL/Z to send):
This is a demonstration message.
^Z
 lougheed@SU-SIERRA.ARPA -- queued
@
```

The mailsystem will try to deliver the message right away. If the foreign host is down, the mailsystem will keep trying periodically for up to three days. If it is unsuccessful at the end of that time, it will mail your message back to you. You will also get your mail back immediately if there is no such user on the other system.

People have occasion to send mail to someone who is not on either the SU-Net or the Arpanet. In this case, you need to know about a host that is willing to *relay* your mail to that other network. The procedure for sending mail to the CSNet and the Bitnet computer networks is relatively straightforward. Typical CSNET and Bitnet mail addresses look like so:

```
Tokano%HP-Labs.CSNet@CSNET-RELAY.ARPA

DF.F26%Stanford.Bitnet@UCB-VAX.ARPA
```

There is no set way of determining which hosts will relay mail; you have to ask a knowledgable person. In any event, once you know your message's ultimate destination and the name of the host that will relay your message, you form the "To: " as in the following example. Here we want to send a message to someone on MIT's local network, the Chaosnet.

```
To: cstacy%mit-oz@mit-mc
```

MIT-OZ is the destination host and MIT-MC is the relay host. If you need to, you can have several layers of relay hosts, each separated by a "%".

If you have accounts on several machines, it is possible to have your mail *forwarded* to a particular host. Contact the system staff of the host from which you wish to have your mail forwarded for details on how to set up mail forwarding.

Bear in mind that you may need special access privileges to send network mail. For example at LOTS, as of this writing, you can exchange mail with people on any of the Stanford machines, but you can not send mail to Arpanet hosts outside of Stanford.

26.5. The FTP Program

FTP is a program that uses either the TCP/IP or PUP suite of protocols to perform various manipulations of files on computers connected by a network. "FTP" is an acronym for *File Transfer Protocol.* Using FTP one can retrieve files from another system (hereafter referred to as *remote files* when it would not otherwise be clear whether they are local or remote), copy *local files* (files in the filesystem of the computer on which you are running FTP) into remote files, rename files, delete files, and display information about files. The program PUPFTP is an an earlier version of FTP that used only the PUP protocol suite; it has been superseded by FTP.

FTP will choose a protocol family to use depending on what protocols the remote computer understands and which protocols FTP thinks will be faster. It is possible, though not usually necessary, to tell FTP which protocol family to use. In the following examples, FTP is using the PUP protocol family. The messages from FTP are slightly different when the

TCP/IP family is used, but the FTP commands are the same.

26.5.1. Basic Use of FTP

To use FTP, type FTP to the EXEC. FTP will print out a message with its name and version number, and then the FTP prompt:

```
@ftp
Stanford TOPS-20 FTP 3.0, type HELP if you need it.
FTP>
```

You will need to tell FTP which computer you want to connect to and what your username is on that host. You use the FTP commands OPEN and LOGIN commands to do this. When you open a connection to a foreign host, it will reply with a message telling you its name and some other information.

```
FTP>open score
 < Score Pup FTP server 2.0 23-Apr-84
Setting default transfer type to Paged
FTP>login kronj
Password:
```

Now you can send and retrieve files. These operations are done with the SEND and GET commands. For either command, type the name of the source file, that is, the file you are sending or getting. Follow this by the name of the destination file, that is, where you want to put the file. FTP will type the names of the files again, just like the EXEC COPY and RENAME commands.

```
FTP>send finger.plan foo.bar
 FINGER.PLAN.1 => <KRONJ>FOO.BAR.3 !! [OK]
FTP>get bar.baz bar.score
 <KRONJ>BAR.BAZ.4 => BAR.SCORE.1 !!! [OK]
```

When you have finished transferring files, leave FTP and return to the EXEC with the EXIT command. This will automatically close your network connection, so if you CONTINUE the FTP program you will have to re-open a new connection.

```
FTP>exit
@
```

FTP will sometimes give you a prompt other than its usual "FTP>". This means that it needs confirmation for some action it is about to take, or that it needs some more information to complete an action. If you don't want to supply this information, or if you don't want the action to be confirmed, type CTRL/G. CTRL/G will also abort multiple-file transfers, but it will *not* stop the transfer of an individual file.

26.5.2. Some Useful FTP Commands

In addition to the OPEN, LOGIN, SEND, GET and EXIT commands, there are a number of other FTP commands for manipulating your network connection and foreign files.

The BYE command closes your current Ethernet connection. To do any more remote file operations you will have to open a new connection.

The CONNECT command tells the remote host that you want full access to a specified foreign directory, and to assume that any filenames without specified directories must be in that directory. FTP will prompt for a password on a separate line; if you don't think you need a password to access that directory, just press RETURN.

The DELETE command asks the remote host to remove a file from its filesystem. There is no FTP command to expunge a directory, nor is there one to undelete a deleted file.

The DIRECTORY command tells FTP to find out the names of all files matching a given specification and type them to the terminal. This command has a number of subcommands.

The HELP command causes FTP to type out information on how to use the program. Type HELP followed by RETURN for examples of basic use of FTP. For help on an individual command, type HELP followed by that command name. The various subcommands of the SET command all have individual help available; type HELP SET followed by a subcommand name to find out about that subcommand.

The RENAME command lets you give a remote filename a different name. Type the current name of the file, followed by the file's new name.

The SET commands set various parameters for the FTP program. Some of these parameters affect file transfers, and others only affect FTP operation. To find out which parameters are set, use the SHOW command. If a parameter is not set and it is needed for a file transfer, FTP may prompt you for it.

The SET CONFIRMATION command makes FTP ask you to confirm what you're doing (press RETURN) whenever it thinks you're about to perform some dangerous action, such as deleting a file.

The SET KEEP command tells FTP to keep certain properties of files in a given type of file transfer. This is useful for maintaining such properties as the last writer, last write, file protection, and so forth.

The SET LOWERCASE command causes FTP to send lowercase filenames. This is very useful when sending files to a computer running the UNIX operating system. UNIX systems are notorious for insisting on lower case filenames.

The SET STATISTICS command makes FTP tell you more information about each file transfer than you can get just from the exclamation marks it always prints. Currently SET STATISTICS types the length of the transfer, the total time taken, and the transfer rate in bits per second.

The SET TYPE command is used to tell FTP what type of transfer mode should be used to send the file over the network. Usually FTP can figure out what the correct mode is. There are three basic transfer modes:

- *Text* mode is used for normal text. Most files are text files.
- *Binary* modes is used for specially formatted files, such as EXE files.
- *Paged* mode is used between DEC-20's for efficient transfers.

For further details on the above and other FTP commands, consult the FTP manual, usually stored as DOC:FTP.DOC.

26.5.3. FTP.INIT Files

When FTP starts up it will automatically look for a file in your login directory called FTP.INIT, and if it finds it, will read the commands in that file. This is useful for setting parameters such as PRESERVATION or STATISTICS, and for telling FTP your username at the various sites with the SET USER command. Below is a sample FTP.INIT file.

```
set statistics                    Print transfer statistics for each file
set keep all all                  Keep all file properties during transfers
set user Sierra Kronj             Login as Kronj on Sierra
set user IOTSA K.Kronj            Login as K.Kronj on IOTSA
```

Below is an example of how FTP uses the SET USER command. Notice that we can type the host name on the same line as the FTP command.

```
@ftp sierra
 < Sierra Pup FTP Server 2.0 25-Aug-84
Setting default transfer type to paged.
Default login as Kronj
Password:                         <-- Type the password here
FTP>
```

26.5.4. The ANONYMOUS Username

Many hosts have an ANONYMOUS username which you can use if you don't have an account on that system. Typically any password will be successful for logging in as ANONYMOUS, but you will be restricted to examining publicly-readable files.

26.6. Other Network Programs

The FINGER command can also be used to "finger" users on foreign hosts. Not every host supports the FINGER protocol, so don't be surprised if fingering users at certain hosts never works. The following command will show everyone logged onto a foreign host, in this case the Score DEC-20.

```
@finger @score
```

To finger a particular user, use this form of the FINGER command:

```
@finger bosack@score
```

It is possible to use the SEND program to send a terminal message to a user logged in on another host. For example, suppose user LOUGHEED on Sierra sent the following message to a user BOSACK on Score:

```
@send bosack@score  Is George ready with the PC version of 10MB board?
```

The recipient would get the following message on his terminal:

```
LOUGHEED@SU-SIERRA.ARPA, 30-Aug-84 4:43PM
Is George ready with the PC version of the 10MB board?
@
```

27. Using Pascal and Passgo

Pascal is an ALGOL-based language designed by Niklaus Wirth in the late 1960's. It incorporates many features of other programming languages and yet is simple for beginners to learn. The Pascal compilers used at Stanford provide enough useful extensions that serious system software can be written in Pascal. Pascal is the language used by the introductory (as well as many upper-division) computer science courses at Stanford.

The language was first specified by Wirth in 1968, and was updated in 1973 and 1978. In 1981 the International Standards Organization (ISO) specified a new standard for Pascal. The ISO Standard defines a new construct called a *conformant array parameter* that will greatly increase the practical applications of Pascal.

The Pascal compiler used at Stanford was developed by Charles Hedrick at Rutgers University. Two versions of his compiler exist at Stanford, called "Pascal" and "Passgo". Pascal is a standard DEC-style compiler which generates object (REL) files to be loaded into core. The Passgo compiler is a local modification of that compiler. It is a load-and-go compiler, generating code directly into memory. The Passgo compiler has the advantage of speed. Because it doesn't save the results of compilation in an external file and because it doesn't load that file into core, it performs much faster on small to medium sized programs. REL files (see page 49) can sometimes complicate the life of beginning students and do take up disk space. For these reasons Passgo is often the recommended compiler.

The Pascal compiler, though slower and a bit more tedious to use, has advantages of its own. It has more available memory for the allocation of variables and activation records. It supports separately compiled modules and linkage to non-Pascal (e.g., FORTRAN) subroutines. The Pascal compiler is the more modern of the two compilers. The Passgo compiler is based upon the 1978 specification of Pascal, whereas the Pascal compiler is based upon the ISO specification and includes conformant arrays.

Many of the properties of the compilers are the same and will be discussed in a single section. The individual properties of each are discussed in the separate sections.

27.1. Documentation

This chapter contains hints on how to write and run a Pascal program. It is oriented towards users who already know Pascal, but who are not familiar with the version of Pascal at Stanford. Other documentation available includes:

- Jensen, Kathleen and Niklaus Wirth, *Pascal User Manual and Report*, Springer-Verlag, 1974. . Contains the standard definition of Pascal prior to 1981. Available at the Stanford bookstore.

- ISO, BS6192,1982. Contains the technical content of the ISO Standard. It can be found in the book *A Practical Introduction to Pascal*, by Addyman, chairman of the ISO committee.

- Cooper, Douglas, *Standard Pascal User Reference Manual*, Norton, 1982. A reference book on the new ISO specification of Pascal. It is not itself the standard, but is much more readable than the standard. Hedrick is basing all of his new documentation on this book.

- Hedrick, Charles, *Pascal-10 and Pascal-20 User's Guide* and *Pascal-20 Hacker's Guide*. The principal documentation for Hedrick's compiler. Copies can be found in the documentation rack at the LOTS Computer Facility in CERAS or online in the DOC: directory.

- Hedrick, Charles, *DEC-20 Pascal and Passgo at Stanford*. Contains detailed information about the compilers used at Stanford. It contains selected versions of the above two documents, slightly modifed and expanded by Stuart Reges. It is also available at the Stanford Bookstore, and in the Computer Science Department Library.

Users without previous experience programming in Pascal may wish to consult one of the standard textbooks on the subject such as *Oh! Pascal!* by Cooper and Clancey.

27.2. Creating and Running your Program

The first step in running a Pascal program is to put your program in a file on the system. To do so, you would normally use an editor such as EDIT, EMACS, TVEDIT or ZED.

As with all types of files, this program file will have a name part, an extension and a generation number. The name part must be six characters or less. The extension should be ".PAS" if you wish to use the Pascal compiler and ".PGO" if you wish to use the Passgo compiler. The system keeps track of the generation number for you. Thus, you might use the file name FOOBAR.PAS for a Pascal program. In general you need not worry about the generation number.

Once you have created a program file using an editor, you can try to run your program. There are three steps involved in running a program: compile the program file; load the resulting machine language translation; and start the program. For a Pascal program, each of these steps is distinguishable. The EXEC commands COMPILE, LOAD, and START carry out the steps. The LOAD command will compile the program if necessary. There is a fourth command called EXECUTE that performs all three steps. You will usually use only the EXECUTE command.

For a Passgo program these steps are all combined into a single action. The COMPILE, LOAD and EXECUTE commands all do the same thing for a Passgo program. The START command will repeat program execution after one of these commands has been used. Again, you will usually use just the EXECUTE command.

For example suppose you create a file called FUN.PGO with these lines:

```
PROGRAM Wow (OUTPUT);
BEGIN
    WRITELN ('This is a simple program');
    WRITELN ('that generates some output.')
END.
```

If you use EDIT as your editor, you will have line numbers:

```
00100   PROGRAM Wow (OUTPUT);
00200   BEGIN
00300       WRITELN ('This is a simple program');
00400       WRITELN ('that generates some output.')
00500   END.
```

You can run this program with the EXECUTE command:

```
@execute fun.pgo
Stanford LOTS/Passgo 20 [WOW    ] -- 1..
Runtime:    0: 0.188
[WOW    execution]
OUTPUT      :       <--- Press the RETURN key here to indicate that output should go to the terminal
This is a simple program
that generates some output.
@
```

You would have gotten the same result if you had used the COMPILE or LOAD commands. You also would have gotten the same result if you has specified just the file name FUN, that is, with an extension. The computer figures out that the .PGO file should be executed with the Passgo compiler. In order to execute the same program with the Pascal compiler, you can rename it:

```
@rename  fun.pgo  fun.pas
 FUN.PGO.1 => FUN.PAS.1 [OK]
@
```

Because the file now has the .PAS extension, the Pascal compiler will be used. You could first compile it:

```
@compile fun
PASCAL: WOW

@
```

Note that only the name portion of the file specification was given. The DEC-20 figures out the extension. The compiler prints the name of the program, WOW, as it works. Note that this name is the Pascal program name from the program

header, not the file name. If you were to look at your directory at this point, you would find a file FUN.REL containing the machine translation resulting from the compilation. After compiling you could load the file:

```
@load fun
LINK:   Loading

EXIT
@
```

and then start it:

```
@start
OUTPUT      :          <--- Press the RETURN key to indicate output to the terminal
This is a simple program
that generates some output.
@
```

All of these steps would have been carried out by the EXECUTE command automatically. The LOAD command is really only useful in conjunction with the SAVE or CSAVE commands that save the core image as an EXE (directly executable) file. Whenever a REL file exists that is more recent than the PAS file, the EXECUTE, and LOAD commands will use the REL file automatically. There is no need to repeat the process of compilation. So, if you were to use EXECUTE at this point, this would happen:

```
@execute fun
LINK:   Loading
[LINKXCT WOW execution]
OUTPUT      :          <--- Press the RETURN key
This is a simple program
that generates some output.
@
```

Notice that the first operation peformed is a load. If you wanted to force the file to be compiled again, you would have to delete the REL file first:

```
@delete fun.rel
 FUN.REL.1 [OK]
@execute fun
PASCAL: WOW
LINK:   Loading
[LINKXCT WOW execution]
OUTPUT      :          <--- Press the RETURN key
This is a simple program
that generates some output.
@
```

The "PASCAL: WOW" at the beginning indicates that the program is being compiled. An annoying situation arises when you are using the Pascal compiler and want to see errors listed twice. Suppose you have a file called BAD.PAS with these lines:

```
PROGRAM Error (OUTPUT);
BEGIN
    WRITELN ('This is a program')
    WRITELN ('with a missing semi-colon')
END.
```

The first statement is not separated from the second by a semi-colon, so a compilation error arises when you try to execute the program:

```
@execute bad
PASCAL: ERROR
    4       WRITELN ('with a missing semi-colon')
P*  1**             ^*****************************
1.^:  Previous statement must end with ";","END","ELSE"or"UNTIL"
?error detected
LINK:   Loading
[LINKNSA No start address]

EXIT
@
```

The compiler encounters a mistake on line 4: line 3 should have ended with a semi-colon. Even though errors were encountered, it tries to load and start the program. Both will fail because the original program was faulty. But now you have a bad REL file called BAD.REL. If you want to see your compilation errors a second time, and try the EXECUTE command again, this happens:

```
@execute bad
LINK:   Loading
[LNKNSA No start address]

EXIT
@
```

Because there is a REL file sitting around that is more recent than the program file, Pascal does not compile the program again. The load and start operations again fail. If you try to COMPILE again, the computer will do nothing because of the REL file. If you wanted to see the listing of errors again, you would do the following:

```
@delete bad.rel
 BAD.REL.1 [OK]
@execute bad
PASCAL: ERROR
    4       WRITELN ('with a missing semi-colon')
P*  1**           ^*****************************
1.^:  Previous statement must end with ";","END","ELSE"or"UNTIL"
?error detected
LINK:   Loading
[LNKNSA No start address]

EXIT
@
```

An experienced user would have used COMPILE instead of EXECUTE if he or she knew the program was faulty and just wanted to see the errors. The new Pascal compiler makes such repeated compilations unnecessary, however. If your program generates errors during compilation, a file with the extension .ERR will be generated with the listing of errors. So after your first execution above, you would have found a file on my account called BAD.ERR with the following lines:

```
    4       WRITELN ('with a missing semi-colon')
P*  1**           ^*****************************
1.^:  Previous statement must end with ";","END","ELSE"or"UNTIL"
```

The error file contains a listing of all program errors. Still, users often encounter strange problems with REL files that lead to the strange "No start address" error message. Deleting the REL file often clears up any confusion.

If you want to get a numbered listing of your program with all of its errors, you can compile or execute it using the "/LIST" switch. If you give the command:

```
@execute bad/list
```

A numbered listing of the program and any error messages will automatically be sent to the printer. If you are working on a computer with no line printer attached to it, the listing will probably go to a file on your directory with the extension LST.

If you are dissatisfied with the indentation or format of your Pascal program, a program called PFORM exists to reformat ("prettyprint") it.

27.3. General Notes on Pascal

The following discussion will be true for both Pascal and Passgo files. The properties specific to each compiler are discussed in succeeding sections.

27.3.1. Pascal Input and Output

The file variables INPUT and OUTPUT are built-in, or predeclared, as in the standard. They don't have to be included in a VAR declaration. The file variables TTY and TTYOUTPUT are also predeclared and represent, respectively, terminal input and terminal output. They work somewhat differently in each compiler, and so are described in the following sections. According to the standard, if you wish to use other file identifiers in your program, you must explicitly declare them via the VAR declaration. They should either be of type TEXT or FILE OF <something>.

If you include a file identifier in the program header, you will be prompted at execution time for a file specification. The user has the option of pressing the RETURN key instead of typing a file name, in which case the default will be used. INPUT and OUTPUT default to the terminal (TTY:). All other file identifiers default to a file by that name on the connected directory with a null extension. TTY and TTYOUTPUT should not be included in the program header since they always represent the terminal and need not be specified.

INPUT and OUTPUT are automatically opened if they are mentioned in the program header and TTY and TTYOUTPUT are always opened if they appear in a program. If you use any other file identifier, you must open it by RESET (for reading) or REWRITE (for writing). For example, if you had a program where you wanted input from the terminal, output to the terminal, input from an external file, and output to an external file, you could use INPUT, OUTPUT, and two file identifiers that you make up, like "InFile" and "OutFile". You would have a program header that looked like this:

```
PROGRAM DoStuff (INPUT, OUTPUT, InFile, OutFile);
```

and you would include the VAR declaration:

```
VAR InFile, OutFile: TEXT;
```

as well as the statements:

```
RESET (InFile);
REWRITE (OutFile);
```

somewhere towards the beginning of the program. If you were to execute a program with the above declarations, you would specify the files at runtime as follows:

```
@execute stuff
Stanford IOTS/Passgo 20 [DOSTUF] -- 1..
Runtime:   0: 0. 79
[DOSTUF execution]
INPUT      :              <--- press the RETURN key to default to the terminal
OUTPUT     :              <--- same as above
INFILE     : stuff.dat    <--- InFile to come from the file STUFF.DAT
OUTFILE    : stuff.out    <--- OutFile to go to the file STUFF.OUT
```

The RESET and REWRITE for INPUT and OUTPUT happen right after the file names are prompted for. REWRITE deletes any old copies of the file and prepares to generate a new one. If you specify a non-existent file for INPUT, it will report that no such file exists and will ask you for another file specification. When an explicit RESET and REWRITE statement is given, it will be executed whenever the flow of control reaches it. If a non-existent file is used in such a RESET statement, then the same kind of error occurs (only this time in the middle of program execution) and the computer pauses and waits for the user to provide a legal file name. If the program terminates abnormally, the output files will not be closed unless the user gives the EXEC command REENTER.

Another way to match file identifiers to actual files is by providing a more complicated RESET or REWRITE statement. You can provide a second parameter that specifies a file name, as in:

```
RESET (INPUT, 'fun.dat');
REWRITE (OUTPUT, 'fun.out');
```

In this case you wouldn't include either INPUT or OUTPUT in the program header.

When you write interactive programs, you don't want the user to have to press the RETURN key at the beginning of

program execution. You would, therefore, not put any file identifiers in the program header and instead include the lines:

```
RESET (INPUT, 'tty:');
REWRITE (OUTPUT, 'tty:');
```

to set up terminal I/O without prompting the user.

This compiler has a special interactive mode for input files. You can specify an interactive file by putting a ":/" after its name in the program header, as in:

```
PROGRAM TerminalFun (INPUT:/, OUTPUT);
```

This will open INPUT in interactive mode. It will still prompt the user at runtime for a file name. If the user specifies a file other than the terminal, then the program will do very strange things. Thus, it is not a terribly useful facility. Another way of specifying an interactive file is to include an optional third argument that specifies the /I switch:

```
RESET (INPUT, 'tty:', '/i');
```

Interactive mode is an obsolete feature of the compiler that Hedrick is trying to phase out. It is discussed on page 184.

There are many other switches that can be specified as the third argument in the RESET statement. For example, you could use:

```
RESET (INPUT, 'ps:<b.bander>foo.dat', '/e/o/u');
```

to open INPUT to the file FOO.DAT in the directory <B.BANDER> on the structure PS:. The switch "/E" makes end of line characters visible; "/O" allows the program to handle file opening errors; and "/U" causes the file to be opened in strictly upper-case mode. Such a command is obviously very complex and should probably only be attempted by users with special requirements. These switches are explained in detail in *Using Pascal and Passgo at Stanford.*

According to the standard, EOLN and the buffer variable (INPUT∧) are undefined when EOF is true. In this compiler, EOLN is TRUE and INPUT∧ is a null (CHR (0) or ∧@) when EOF is TRUE.

It is an error for a program to execute a READ or READLN statement after an end of file has been reached. The Pascal compiler at Stanford does not detect an error on such a condition. Instead it returns garbage values and continues program execution. Be careful!

The compiler does detect other read errors, such as trying to read a numerical value when the next item in the file is non-numeric. When reading from an external file, such errors cause the program to stop execution. When reading from the terminal, the computer reports the error (although not usually in a straightforward manner) and asks the user to try again.

27.3.2. Strings in Pascal

Pascal is not known for good string manipulation. You will find that this new compiler is somewhat helpful in that regard. A variable of type ARRAY OF CHAR is handled very well by the compiler. It can be read in and written out directly and two such variables can be compared for equality, less than, greater than, and so forth. If you give a statement like:

```
READ (name);
```

where NAME is declared as a variable of type ARRAY OF CHAR, then characters will be read from the input file into NAME until either an end-of-line is encountered, or NAME is filled. If the end of line is encountered before NAME is filled, NAMEwill be padded with spaces. If you say:

```
WRITE (name);
```

the contents of NAME will be written out. All of the contents will be written out. This means that even if NAME contains an eight character word with fifty-two trailing spaces, all sixty characters are written out.

A more general form of the READ and WRITE statements is available. You can say:

```
READ (name:length:[':', '!']);
```

Again, this reads into the variable NAME, but it also says to put the number of characters read in into the integer variable LENGTH and to use a break set (in this case, the elements colon and exclamation mark). Characters will be read until until the array is full, end of line is reached, or until one of the break characters is encountered. The break character is not read in. The more general form for output is:

```
WRITE (name:length);
```

which will output NAME with a field width of LENGTH

Comparing for less than or greater than follows the standard rules of alphabetizing. So you can say things like:

```
IF name1 < name2 THEN WRITELN ('Yeah.  Name2 is bigger.');
```

Of course, the two strings have to be of the same type. You can't compare a sixty-character array to an eighty-character array.

All of this holds true of the type PACKED ARRAY OF CHAR as well, with one addition. An example of a PACKED ARRAY OF CHAR is a string constant like `'Hi there'`. The string constant must have as many characters as the array has elements, otherwise you will get a type conflict.

27.4. Terminal I/O in Passgo

Terminal I/O for the Passgo compiler is worth mentioning because it differs so much from Pascal. As mentioned above, TTY and TTYOUTPUT are predeclared in Passgo and Pascal. Actually, you can use TTY for both input and output, but in Passgo the two file variables will share a character buffer. If you understand what that means, you realize that it can lead to messy situations and very difficult bugs; if you don't understand, take it on faith that you shouldn't use TTY for both input and output.

Terminal input in Pascal was first implemented on a batch system, with card input. It is not surprising that it and Passgo do not deal with terminal interaction well.

Passgo does one character look-ahead on input. If you ask in a program whether EOLN is true, for example, you are asking something about the next character without actually reading it in. The buffer variable also performs one character look-ahead on an input file. What happens when input is coming from the terminal? How can you look ahead one character when you don't know what the user will type next? Think about the first line of user input, for example. How can you know what the first character typed by the user will be? If you execute this program:

```
PROGRAM Mystery (INPUT, OUTPUT);
BEGIN
    IF EOLN THEN WRITELN ('Yes, it is true folks.')
    ELSE WRITELN ('No, it is false folks.')
END.
```

and direct both INPUT and OUTPUT to correspond to the terminal, what output will the program produce?

The problem with this program is that it asks what the user will type before the user has typed anything. The solution arrived at by Passgo is very strange. This compiler has the convention that if, at any time during program execution, the buffer variable (i.e., the next character to be read in) is undefined, then the program will pause and wait for the user to type an entire line of input.

At the beginning of program execution the buffer variable is undefined. What does the program do? It issues a request for input before executing the program. Here is the execution of that program in Passgo:

```
@execute strange
Stanford LOTS/Passgo 20 [MYSTER] -- 1..
Runtime:   0: 0.227
[MYSTER execution]
INPUT      :       <--- press the RETURN key to default to the terminal
OUTPUT     :       <--- press the RETURN key to default to the terminal
[INPUT, end with ^Z: ]
                   <--- press the RETURN key as your line of input
Yes, it is true folks.
@
```

After the program prompts you for the names of INPUT and OUTPUT, it prompts you for a first line of input and pauses until you type something. When it says to end with "^Z", it is telling you that you can indicate the end of file by typing CTRL/Z. You never really want to do that when reading from the terminal. In this case, you press the RETURN key right away. This made EOLN TRUE initially, and the program wrote out its "yes" message. If instead you had typed something on that line, then EOLN would have been FALSE initially and the program would have generated its "no" message.

Interactive mode offers something of a solution. Above it was described how a file can be opened in interactive mode. Note that the TTY is automatically opened in interactive mode. When a file is opened in interactive mode, its buffer variable is initialized to a null (CHR (0) or ^@). EOLN is TRUE in this case. This avoids the problem of the buffer variable being undefined before program execution begins. Thus, if the above program had used TTY and TTYOUTPUT instead, it would not have prompted the user for a line of input and would have printed the "yes" response automatically.

It is good practice when reading from the terminal (either via INPUT or TTY) first to use WRITE or WRITELN to send a prompt to the terminal, then use a READLN without any variables, then a READ statement with the desired variables. This ensures that you are always looking at the end of the previous input line. The following is an example of a function that is frequently used with terminal I/O:

```
FUNCTION Yes: BOOLEAN;
(* asks the user yes or no and returns TRUE for yes *)
VAR ch: CHAR;
BEGIN
    REPEAT
        WRITE (TTYOUTPUT, 'Yes or No>');
        READLN (TTY);
        READ (TTY, ch)
    UNTIL (ch = 'y') OR (ch = 'n') OR (ch = 'Y') OR (ch = 'N');
    yes := (ch = 'y') OR (ch = 'Y')
END;   (* yes *)
```

27.5. Terminal I/O in Pascal

The interactive mode mentioned in the previous section can be used in the Pascal compiler as well, although it is not needed. The Pascal compiler has a much better facility for terminal interaction. The file variable TTY is not automatically opened in interactive mode. It is opened in *lazy I/O mode* instead.

The lazy I/O solution to the problem of one character look-ahead is to wait until absolutely necessary to do the look-ahead. Evaluating EOLN or the buffer variable (INPUT^), performing a READ or GET, and passing the buffer variable as a parameter all require the one character look-ahead. In true lazy I/O, this look-ahead is performed only when the value is needed and not before (it is "lazy" about doing the work and procrastinates until the last moment).

In the case of the program mentioned in the previous section, the first statement of the program evaluates EOLN, which means one character look-ahead is necessary. So the Pascal compiler exhibits the same behavior as the Passgo compiler for this program: it pauses right away and waits for the user to type a line of input. It doesn't provide the "INPUT, end with ^Z: " prompt, however. Below is an example of a program with a much better design.

```
PROGRAM Add (INPUT, OUTPUT);
VAR first, second: INTEGER;
BEGIN
    WRITELN ('I will add together two integers for you.');
    WRITE ('Give me two integers to add, please ---> ');
    READLN (first, second);
    WRITELN (first, ' + ', second, ' = ', first + second)
END.
```

With lazy I/O, we don't need to perform one character look-ahead until we reach the third statement, the READLN. The first two statements can be executed without prompting the user for input. After we have produced some output prompting the user, the read operation takes place. Here is a sample execution:

```
@execute add.pas
PASCAL: ADD
LINK:   Loading
[LNKXCT ADD execution]
INPUT      :          <--- hit the RETURN key to default to the terminal
OUTPUT     :          <--- same
I will add together two integers for you.
Give me two integers to add, please ---> 45 987
          45 +          987 =          1032
@
```

The program behaves as expected. It issues the initial prompt and then pauses for a line of input.

The Rutgers Pascal compiler has partial lazy I/O in that the READLN operation is lazy. The first character of a line of input is not read from the terminal until one character look-ahead is required. Once a line of input is begun, the computer pauses until the user types the whole line of input.

The proper protocol for terminal interaction in the Pascal compiler is to use WRITE or WRITELN to send a prompt to the terminal, then READ statement(s) that process the line followed by READLN. Thus the function YES given above would be written in Pascal as:

```
FUNCTION Yes: BOOLEAN;
(* asks the user yes or no and returns TRUE for yes *)
VAR ch: CHAR;
BEGIN
    REPEAT
        WRITE (TTYOUTPUT, 'Yes or No>');
        READLN (TTY, ch)
    UNTIL ch IN ['y', 'n', 'Y', 'N'];
    yes := ch IN ['y', 'Y']
END;   (* yes *)
```

27.6. External Declarations in Pascal

As mentioned earlier, the Pascal compiler supports linkage to external routines. As a simple example, consider the non-standard Pascal function RANDOM (0). RANDOM returns a random number greater than or equal to 0.0 and strictly less than 1.0. The problem with this function is that it gives the same sequence of random numbers every time. This is not terribly interesting for game playing programs and the like. There is a FORTRAN procedure called SETRAN that can be used to truly randomize the random number generator used by RANDOM. SETRAN takes one argument, a "seed value" for the generator. If you want to include SETRAN in a Pascal program, you will have to declare it as a FORTRAN procedure, as shown below:

```
PROCEDURE SetRan (argument: REAL); EXTERN FORTRAN;
```

The EXTERN FORTRAN takes the place of the procedure body, so no declarations are needed and neither is the BEGIN/END block. You can also use the non-standard variable TIME which returns the time of day in milli-seconds. To set up the random number generator, include the statement:

```
SetRan (time);
```

somewhere towards the beginning of the program. This will provide you with a different sequence of random numbers every time you run the program. The protocol for writing separately compiled modules is complicated enough that it is

not discussed here. Interested users are directed to *DEC-20 Pascal and Passgo at Stanford.*

27.7. Conformant Array Parameters

As noted earlier, the Pascal compiler supports a new feature of Pascal call *conformant array parameters.* Pascal requires that you pass parameters of a named type. You cannot have a procedure with a parameter like this:

```
PROCEDURE Foobar (arg: ARRAY [1..10] OF INTEGER);
```

You must give a name to that type with a TYPE declaration and then pass it as a parameter as in this example:

```
TYPE list = ARRAY [1..10] OF INTEGER;

VAR OneList: list;

PROCEDURE Foobar (arg: list);
```

In this case `OneList` can be passed as a parameter to `Foobar`. This causes a problem, however, when you want to perform the same operation on two different types of arrays. For example, suppose you had these declarations:

```
TYPE list100 = ARRAY [1..100] OF INTEGER;
     list500 = ARRAY [1..500] OF INTEGER;

VAR OneList: list100;
    TwoList: list500;
```

If you wanted to sort these lists into numerical order, you would need two procedures

```
PROCEDURE Sort100 (VAR TheList: list100);
. . .
PROCEDURE Sort500 (VAR TheList: list500);
```

where the only difference was the length of the array. You can avoid such redundancy by using a conformant array parameter:

```
PROCEDURE Sort (VAR TheList: ARRAY [low..high: INTEGER] OF INTEGER);
```

This defines a sorting procedure that takes as a parameter any array indexed over integers that contains integers. The identifiers "`low`" and "`high`" represent local constants that are set by the procedure. You cannot change their values, nor can you pass them as parameters (that is, you can't specify that only a portion of an array be passed). If you were to make the procedure call:

```
sort (OneList);
```

`low` would be automatically set to 1 and `high` would be set to 100, because `OneList` is an array [1..100] of integer.

The general syntax of the conformant array parameter is:

ARRAY [*identifer..identifer* : *ordinal type*] OF *type*

A major flaw in using Pascal has been the lack of a good string package or a general-purpose set of matrix operations. With the conformant array parameters it is now possible write such general-purpose procedures and functions in Standard Pascal. With the facility for writing separately compiled modules in this compiler, it is now possible to create a Pascal library of routines that can be invoked via EXTERN declarations.

For further information on conformant array parameters, read Doug Cooper's book mentioned on page 177. For more information on separately compiled modules, consult *DEC-20 Pascal and Passgo at Stanford.*

27.8. Debugging a Pascal Program

Pascal and Passgo programs may be run with the Pascal Debug System (PASDDT) by using the EXEC command DEBUG instead of the EXECUTE command. Successful debugging also requires that the program be assembled with /DEBUG, but this happens by default at Stanford. For example, the following command will cause the program FOOBAR to be run under the control of PASDDT:

```
@debug foobar.pgo
```

PASDDT is an extra command interpreter that allows you to control the execution of your program. It lets you stop your program at places that you have specified in advance (breakpoints). When PASDDT is in control, you can look at the current value of the variables in your program (using normal Pascal notation), and you can even assign new values to them.

27.8.1. How PASDDT works

PASDDT commands can only be given when PASDDT is in control. When you start out debugging a program, PASDDT is initially in control. PASDDT continues accepting commands until you tell it to start your program. Once you have done that, your program is in control until something happens to return control to PASDDT. Here are the ways that PASDDT can regain control:

- You can set *breakpoint*. This is a specific line in your program where you want to look at variables or give PASDDT commands. When the executing program reaches such a line, it stops and returns control to PASDDT. You insert breakpoints by using the STOP command, described below.
- You can request *single-stepping*. This allows you to execute one line of the program at a time, with PASDDT regaining control after each line is executed. You enter single-step mode with the STEP command.
- When an error occurs, control goes to PASDDT, allowing you to examine the point in the program where the error occurred.
- If you need to get into the debugger while your program is running, type CTRL/D. This will cause your program to stop executing and pass control to PASDDT, no matter what your program happened to be doing at the time.

Once PASDDT has regained control, you can look at and change variables, and give other PASDDT commands. Then you can request PASDDT to continue your program. Your program will then continue from where it was when you stopped it.

Many PASDDT commands use line numbers to refer to a line in your program. If your file contains EDIT line numbers, these line numbers are used. Otherwise, 1 refers to the first line, 2 to the second, and so forth.

Many files are organized into pages. In EDIT, the M command is used to produce page marks. In other editors, pages are delimited by form feed characters (CTRL/L). In PASDDT you can specify line 75 on page 3 with the command "75/3". If you do not specify a page number, it is assumed that you are referring to the current page (the page on which the most recent typeout of program text began). Line numbers start over again with 1 on each page, except in EDIT files, where the EDIT line numbers are used.

Here are some examples of legal line number specifications.

1864	line 1864 on the current page
1200/5	line 1200 on page 5
*	the current line (where the most recent typeout of program text started)

You can find out what the current line and page are by giving the PASDDT command "*=".

27.8.2. Commands for Controlling your Program

After you give the DEBUG command, your program will go through the normal startup dialog. You will be asked for file names for all files listed in the PROGRAM statement. If INPUT is assigned to the terminal (and is not declared interactive), you will be asked for the first line of input from the terminal. Once this is finished, PASDDT will get control. You will see something like the following:

```
@debug triang.pgo
Stanford LOTS/Passgo 20 [TRIANG] -- 1.. (1000)
Runtime:   0: 0.330
[TRIANG execution]
OUTPUT     :
> Stop at main BEGIN - module TRIANGLE open at 600/1
00600       FOR count1 := 1 TO max DO values [count1] := 0;
00700       values [1] := 1;
00800       FOR count1 := 1 TO max DO
>>
```

The ">>" prompt shows that PASDDT is waiting for a command. PASDDT always shows you the line of your program that will be executed next if you continue the program. In this case it is line 600 on page 1 of your program. A message like this will be printed whenever PASDDT gets control from your program.

The following commands can be used to specifiy when PASDDT will next get control of your program's execution:

`STOP line-number`
: This puts a breakpoint at the specified line. If you continue your program's execution, and if the program ever reaches this line, it will be suspended and PASDDT put in control. You will then see a message similar to the one shown above. The ">>" prompt will tell you that PASDDT is again waiting for commands.

`STOP LIST` Lists the line numbers of all the breakpoints.

`STOP NOT line-number`
: Removes the breakpoint at the specified line.

`STOP NOT ALL` Removes all breakpoints.

`END`
: This ends the control by PASDDT. It causes your program to proceed from the point where it was most recently stopped. The program will continue until something happens to give control back to PASDDT. Most commonly it will continue until it reaches a line where a breakpoint has been set. If you have been single-stepping, END cancels single-step mode.

27.8.3. Single-Step Mode

Single-step mode is a convenient way of executing your program while maintaining a maximum of control under PASDDT. In contrast to breakpoints, which are used when you know what part of the program you are interested in looking at, single-step mode is used when you don't know exactly what you are looking for. It causes only one line of your program to be executed. PASDDT regains control at every line.

Here is what you will see at each line when you are in single-step mode:

```
>> step
> Stop in TRIANGLE:1000/1
01000           FOR count2 := count1 DOWNTO 2 DO
01100               values [count2] := values [count2] + values [count1 - 1];
01200           WRITELN;
S>
```

This indicates that the next line to be executed is line 1000 on page 1. The prompt of "S>" instead of ">>" indicates that you are in single-step mode.

Here are the commands that are used to start and stop single-step mode, and to control your program when you are in that

mode:

STEP Start single-stepping. This will cause your program to regain control, but will return control to PASDDT at the next line. This command is valid whether you are in single-step mode or not. It leaves you in single-step mode.

RETURN key When you are in single-step mode, a simple carriage return is equivalent to the command STEP. This is simply a convenience to allow you to move through your program without having to type the word "STEP" for each line to be done. In fact the only difference between single-step mode and normal PASDDT is the fact that in single-step mode, pressing RETURN and ESC keys are convenient shortcuts. All normal PASDDT commands are available in single-step mode.

ESCAPE key When you are in single-step mode, you sometimes find yourself stepping through procedures in which you are not interested. If you press the ESC key, execution of your program will continue until the program exits from the procedure it is currently executing. When the program reaches the next line after the call to that procedure, PASDDT will regain control. You will probably have to try single-stepping before you see why this is a useful feature.

END You get out of single-step mode by continuing your program in the normal way with the END command.

27.8.4. Commands for Looking at and Changing Variables

The main reason for using PASDDT is to allow you look at what is going on inside your program. The commands listed in this section are those that allow you to look at variables and optionally change their values.

variable = This command shows you the value of a variable from your program. There is some problem because of Pascal's block structure. There can be 25 different variables in your program all called "I". Each one is defined in a different procedure. PASDDT uses the same rule for figuring out which one you mean that Pascal itself uses. The search starts at the location that was shown when PASDDT gained control (the next line to be executed). Any variable defined in the procedure of which that line is a part is used first. Then procedures containing that one are searched, until the search ends with global variables.

Note that "variable" means any legal Pascal variable. You can use subscripts, dots, pointer arrows, and even complex combinations of these. If you ask for an array or a structure, all of its elements will be printed. For an array, several identical elements are printed in an abbreviated form.

variable := value This allows you to change the value of a variable. Whatever value you set it to will be used by the program if you continue it. Value should be a Pascal constant of a type compatible with the variable involved.

TRACE TRACE gives you a "backtrace", a list of what procedures are currently active, and where they were called from. If you do not want to see the whole backtrace, you can use the TRACE n form.

TRACE n Prints only the first N procedure calls from the full trace.

STACKDUMP STACKDUMP prints the value of all local variables. Needless to say, this output can get a bit voluminous. It is possible to direct the output of this command to a file by means of STACKDUMP 'filename'.

It is also possible to specify that you only want to see local variables from the N innermost procedures currently active. You can see what these are from the output of TRACE. Use STACKDUMP n.

27.8.5. Looking Around in the Source File

Sometimes you will want to place a breakpoint somewhere, but not remember the exact line number. Or for some other reason you may need to look around in the source file. For this purpose PASDDT has the two following commands.

`FIND 'string'` This command searches forward in the source file, beginning from the current location (denoted by a dot). It looks for the first occurrence of the string specified. In doing the search, upper and lower case letters are considered equivalent. So if you give the command FIND 'MYPROC', this will match an appearance of MyProc in the source file. The first line found will be displayed, and will become the new current line.

`TYPE start-line end-line`
This allows you to type out any part of the source file on your terminal. If you specify only one line number, just that one line will be typed. The ending line number can be something ridiculously big if you want to see everything to the end of the file.

27.8.6. A Warning About Confusing PASDDT

It is possible for PASDDT to become confused about where it is in your program. This will generally happen if you transfer to PASDDT by typing `CTRL/D` in the middle of your program. As long as the part of your program that is written in Pascal is being executed, things will be fine. But if you happen to be in one of the library routines (e.g. I/O routines, NEW, or routines written in another language and loaded with your Pascal program), PASDDT will be unable to figure out where you are. In this case, it will usually claim that you are at either line 0/0, or at the last line in your program.

The only serious problem raised by this case is that you can't tell exactly where you are in the program. The usual functions of PASDDT should still work. Note that the TRACE command will at least tell you what procedure you are in. And END and STEP can still be used to continue your program. In fact, STEP is a convenient way to find out where you are, as it will cause a stop at the next line in the source program.

This problem happens most often when you stop the program while it is waiting for input from the terminal.

27.9. Known Bugs and Deficiencies in Pascal and Passgo

Sets in both compilers are limited to 72 elements. This is a problem considering the fact that the compiler uses the standard ASCII character set of 128 elements. The Passgo compiler does not distinguish upper and lower case in SETs and will ignore many control characters. The Pascal compiler, however, has a special facility that allows it to handle sets of CHAR properly. In addition to the size limitation, the compiler puts limitations on the range of values for set constants with integers in them. You are restricted to the range 0..71. Naturally, you can still define set variables over different ranges provided the range has no more than 72 values in it. The Passgo compiler has a strange bug for set constants with integers in them. As noted, the values of the set constant must be between 0 and 71. The Passgo compiler, when testing an expression for set inclusion, will mod the expression by 256. So, not only will the number 1 be in the set `1..5`, but also 257, 513, 769, etc. This bug does not exist in the Pascal compiler.

Real expressions in both compilers will always be written out with a leading blank. If formatted output is used, the field will be expanded sufficiently for the leading blank. The number 2.3 written out with the format ":3:1" will require expansion in order to include the leading blank. The number -2.3 will be expanded by two columns, one for the sign and one for the leading blank.

In standard Pascal, `EOLN` and the file buffer variable will be undefined when `EOF` is true. When `EOF` is true in this compiler, `EOLN` will be true as well and the file buffer variable is a null (`CHR(0)` or `CTRL/@`). If a program attempts to read beyond `EOF`, the program does not, as expected, reach a fatal error. Garbage will generally be returned.

As noted on page 183, if you use the file identifier TTY in Passgo for reading from and writing to the terminal, they will share a common single-character buffer.

When you pass a procedure or function as a parameter to another procedure or function, you are bound to run into trouble with the Passgo compiler. It is not worth listing all of the bugs. If you want to pass procedural or functional parameters, use the Pascal compiler.

The careless insertion of a back-slash character, "\", will cause a Passgo program to compile indefinitely. This program

```
00100   PROGRAM Bug;
00200   \
00300   BEGIN
00400   END.
```

will compile indefinitely under the Passgo compiler. To find such a mistake, execute the program with the "/PASCAL" switch to force the Pascal compiler to be invoked.

The allocating of local variables for procedures works slightly differently when you are in EXECUTE mode versus DEBUG mode. There are programs that work that should not work because they rely on local variables being allocated in a particular way (e.g., the same values they had during the last call of the same PROCEDURE). Programs like these can cause severe headaches. There are programs that work in the EXECUTE mode, but which reach a fatal error in the DEBUG mode. This kind of program is preferable to the one that doesn't work in the EXECUTE mode but which *does* work in the DEBUG mode (how can you debug a program that works in debug mode?). If you encounter a program that behaves differently when DEBUGed versus EXECUTEd, look for local variables that were not initialized properly.

28. Using PHOTO to Record Your Terminal Session

PHOTO is a program to make a log file which records everything that appears on the screen during the terminal session. This file can be used to generate a hard copy listing of the session on the line printer. It can be edited to be included in some other document, such as class handouts.

To use PHOTO, just type PHOTO to the EXEC. PHOTO will print a help message and ask you what you want to call the log file it will create. If you respond with the name of an existing file, PHOTO will create a new version for this session's log. To leave PHOTO, give the EXEC command @POP (this is the normal command to pop up to the previous command level). In the following example, we use PHOTO to demonstrate the DAYTIME command:

```
@photo
Welcome to PHOTO.  To exit you must type 'POP' when at the EXEC command level.
For more help type 'HELP PHOTO' to the EXEC.

Log File (press RETURN to default to PHOTO.LOG): daytime.log

[PHOTO:  Recording initiated  Mon 24-Aug-84 3:40PM]
@daytime
Monday, August 24, 1984 15:40:12
@pop
[PHOTO:  Recording terminated Mon 24-Aug-84 3:40PM DAYTIME.LOG.1]
@
```

The file DAYTIME.LOG now contains a copy of everything that appeared on the terminal screen between the @PHOTO and the @POP commands. Note: if you are documenting the execution of a program, be careful NOT to call the log file by the same name as your program! If you do, you won't have any program left when you're done. ".LOG" is a good file type for PHOTO log files.

PHOTO checks the characteristics of your terminal once when you start it up, so you should not change them while in the middle of PHOTO. Set any nonstandard terminal characteristics you wish to use before starting PHOTO; most users can just leave them as they are.

If in the middle of logging a session, you come upon a section you do not wish to log, you can suspend logging by typing CTRL/Y. Another CTRL/Y resumes logging.

Warning:

Do not use display programs such as EMACS, SYSDPY, etc. when in PHOTO. Use of display programs with PHOTO can result in the death of your job, or worse.

29. Plotting Packages

The first part of this chapter covers PLOT, a plotting system developed and supported by Ivor Durham at Carnegie-Mellon University, and used at Stanford with permission. This system is generally available on the Stanford DECSYSTEM-20's. The remaining sections deal plotting packages available only at LOTS: OMNIGRAPH (and associated support programs, PLOTX and OMPLOT), H2PLOT, and LOTPLT.

There is also a modeling laboratory program called MLAB that can be used to generate sophisticated plots. See Chapter 25, page 163, for further details.

You can plot with either characters or with dots and vectors. Character plotting requires no special terminal or printer features and can be done by your own programs or by statistical packages such as MINITAB or SPSS. Character plots can be displayed on either the CRT screen or any line printer. They have the advantage of being fast and easy to do. In addtion programs such as MINITAB and SPSS can bring their curve fitting capabilities to bear on the plot. However, if more than one line is to be plotted on a character plot, it is difficult to achieve accuracy because of the lack of resolution.

Dot plotting is done with dots and vectors instead of characters, and requires a graphics output device such as the Printronix line printer, a graphics terminal, or the Dover or Imagen laser printer. Plots obtained with dot plotting are accurate, clear, but are somewhat slow both to generate and to print.

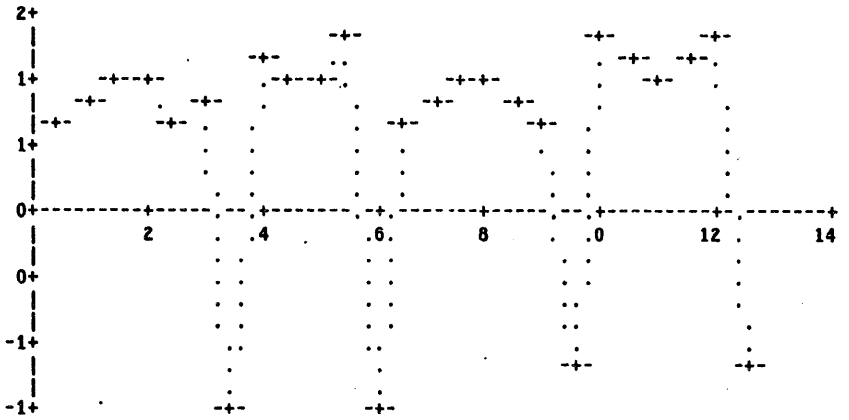
29.1. The PLOT Program

PLOT is a general purpose plotting package. With it you can generate fairly complex plots of your data with annotation, axes, colors, etc. For the novice user PLOT offers a `NOVICE` command that guides the user through a tutorial in the use of PLOT. To obtain documentation on PLOT, see the file PLOT:PLOT.DOC. The file HLP:PLOT.HLP contains updated command descriptions of commands documented in PLOT.DOC. If you want to use a text editor to read PLOT.DOC, you can use the PLOT command `DOCUMENTATION` as shown below.

```
@plot
Plot 25H(664)  Copyright (C) 1984 by Ivor Durham.
Type Help if you need it.

Use the News command for details of recent changes and the Gripe command to
report problems with Plot.  First time users should try the Novice command.

Plot>documentation
```

Below we give a few examples of how to use PLOT. In the first example we read data from a previously generated file, plot it on the terminal, and save the commands in a file for later use. The axes and other parameters are chosen automatically.

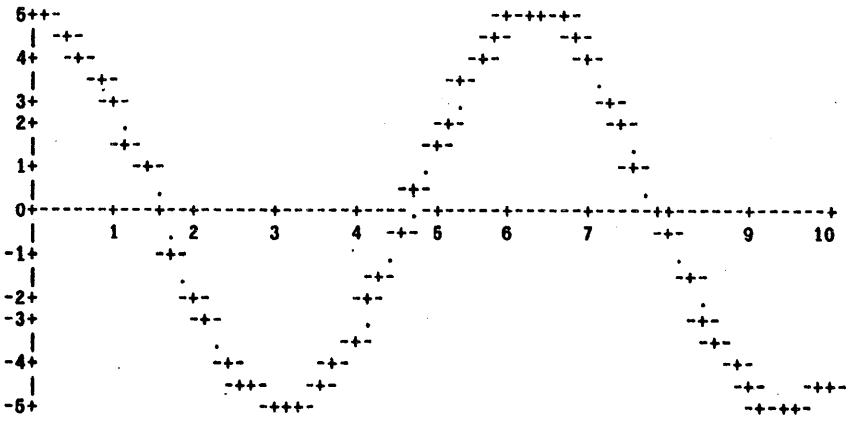
```
@plot
Plot 25H(664)  Copyright (C) 1984 by Ivor Durham.
Type Help if you need it.

Use the News command for details of recent changes and the Gripe command to
report problems with Plot.  First time users should try the Novice command.

Plot>new graph
Plot>new points
Type in points as "x,y {~<Confidence Interval>}<return>"
Finish list of points with an empty line.
001>    @strange.dat                                  <-- The data is in this file.
026>                                                  <-- The user presses the RETURN key here.
Plot>plot
Plotter? [TTY]:                                       <-- The user presses the RETURN key to print the plot on the terminal.
Do you want to Save the graph in a command file? [Yes]:  <-- The RETURN key was pressed.
Command file name [DSK:STRANGE.CMD]:                  <-- The RETURN key was pressed.
```



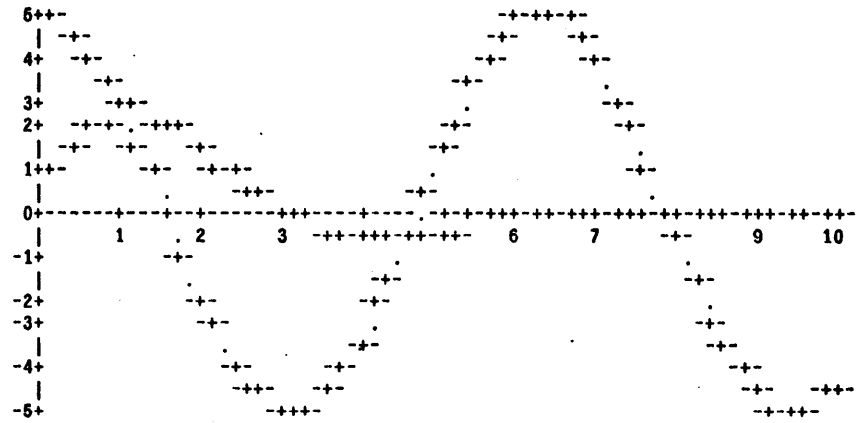
```
Plot>exit
Good-bye.
@
```

In this second example we generate two curves displayed on the same plot. We save the commands in a file for later use.

```
@plot
Plot 25H(664)  Copyright (C) 1984 by Ivor Durham..
Type Help if you need it.

Use the News command for details of recent changes and the Gripe command to
report problems with Plot.  First time users should try the Novice command.

Plot>new graph
Plot>new curve
Plot>add
Type in points as "x,y {~<Confidence Interval>}<return>"
Finish list of points with an empty line.
001>    @sin.dat
051>                                          <-- The user presses the RETURN key
Plot>plot
Plotter? [TTY]:                               <-- Press RETURN to send plot to terminal.
Do you want to Save the graph in a command file? [Yes]: no
```



```
Plot>new curve
Plot>add
Type in points as "x,y {~<Confidence Interval>}<return>"
Finish list of points with an empty line.
001>    @exp.dat
051>                                          <-- RETURN key pressed here.
Plot>plot
Plotter? [TTY]:                               <-- RETURN key pressed here.
Do you want to Save the graph in a command file? [Yes]: yes
Command file name [DSK:EXP.CMD]:              <-- RETURN key pressed here.
```



```
Plot>exit
Good-bye.
@
```

29.2. The OMNIGRAPH Package

OMNIGRAPH is a set of graphics subroutines for interactive plotting on graphics display terminals or printing on a plotting printer such as the Printronix. The routines can be called from SAIL, FORTRAN, or Pascal programs, and can be used with a wide variety of terminals. It is not possible to use OMNIGRAPH with Passgo. The routines are device independent in that the program need not know what kind of a display the program will be run on.

The OMNIGRAPH system provides support for the DEC GIGI, Tektronix 4010, Tektronix 4014, Tektronix 4025 and the Printronix printer.

The OMNIGRAPH system performs 3-D graphics, windowing, viewporting, and rotation operations as primitives. The user need not worry about scaling data points or clipping curves that do not fit the screen; these actions are performed automatically. It is possible to save the current image in a file. Using the two auxilliary programs described below, OMNIGRAPH provides you with the capability of interactively viewing your graphics on a display terminal and then choosing particular plots to be plotted on a (slower) plotting device.

Documentation is available on-line. For a brief description of the most important OMNIGRAPH routines, give the command `HELP OMNIGRAPH`. The OMNIGRAPH manual is available on the bookshelves in CERAS. It includes a brief but helpful tutorial on graphics principles. Persons interested in OMNIGRAPH's implementation are welcome to look at the files `OMNI:OMNI-IMPLEMENTATION.MANUAL` and `OMNI:DISDOC.MEM`. There are also a variety of help files in the `OMNI:` directory.

29.2.1. The PLOTX Program

PLOTX is a program to take `PLX` files generated by the MLAB program or the OMNIGRAPH subroutines and convert them into a format usable by a plotting printer. *Do not* try to print a file with a `PLX` extension without first using PLOTX. PLOTX also allows you to alter the plot before sending it to the plotter. You can rescale the plot in either direction (X or Y), add an offset to the X or Y coordinates, or even overlay several plots.

Run PLOTX by typing the command PLOTX to the EXEC. PLOTX will then ask you several rather self-explanatory questions such as which files to convert, whether you wish to modify the plot, if so, how, and so on. Each time the questions are asked, the "default" answer is shown in brackets beside the question. If the default answers are satisfactory, then all you need do is press the `RETURN` key. PLOTX creates the printable file PLOTX.OUT, unless you give it another output filename to use when asked by the program. You can then use the EXEC `PRINT` command to send the output file to a printer.

29.2.2. The OMPLOT Program

OMPLOT is much less flexible than PLOTX, although simpler and faster. OMPLOT will display a PLX file on a graphics terminal such as a DEC GIGI terminal. The program will allow the user to scale the plot to make sure it fills his particular screen, even if the plot did not do so when created. You run OMPLOT by typing the word OMPLOT to the EXEC.

The program first asks you what type of terminal you are on. If you desire, choosing display type 4, the Printronix printer, will cause OMPLOT to be "fooled" into sending a printable version of the plot directly to the printer instead of to your terminal. When using display type 4, if you ask OMPLOT to scale the plot so that it fills the "screen", OMPLOT will insure that your plot is nearly the full size of the printer page. Note that when you use this trick, there is no output file to print as with PLOTX.

29.3. The H2PLOT Program

H2PLOT is a plotting package for interactively plotting functions or data. It produces character plots to allow you to view your data on the terminal and can generate a dot plot for the Printronix line printer. H2PLOT is similar in spirit to the MLAB modeling laboratory, but suffers from an obscure user interface, poor documentation, lack of powerful features, and a number of bugs, some subtle and others not. If you are deciding which plotting package to learn, use PLOT or MLAB instead.

To run H2PLOT simply type

```
@h2plot
```

The program is broken up into six command modes, each with a different prompt.

Prompt	Mode Name	Purpose
H2PLOT>	Main	Top level of H2PLOT.
P-EXPR>	Expression	Function plotting.
P-DATA>	Data	Data plotting.
P-PARM>	Parameters	Change default plot parameters.
P-D-ARR>	Arrange	Arranging and transforming data.
P-D-FIT>	Fitting	Draw smooth curves through data points.

The following sections summarize the functions and basic commands of each command mode. Following that are two examples of H2PLOT's use. If at any point you don't know what a command does, type HELP followed by the command name.

29.3.1. Top Level H2PLOT Commands

When you start up the H2PLOT program you are automatically put into the H2PLOT top level. At this point your prompt will be "H2PLOT>". The possible commands are listed below.

CHANGES — Lists any recent changes to H2PLOT. The developer of this program left Stanford several years ago, hence there is unlikely to be much activity.

DATA — Puts you into the DATA command mode to enter data to be plotted from a file or the terminal.

EXPRESSION — This puts you into the EXPRESSION command mode. If you have not already entered a function, it will automatically prompt you for one. The capability to edit and save functions does not exist, hence it is best to stick to straightforward functions. For complicated functions write an external program to generate the points and then read them using the DATA command mode.

GRIPE — Allows you to enter a gripe or suggestion about H2PLOT. See CHANGES.

HELP — Gives a general help message for beginning user.

PARAMETERS Puts you into the PARAMETERS command mode. This mode allows you to alter things like the plot's size, values, or axis labels.

PLOTFILE The plots you produce by giving the PRINT command in either the EXPRESSIONS mode or the DATA mode can go directly to the line printer or be stored in a disk file to be printed later. The default is for each plot to go to the line printer. The PLOTFILE command is used to specify the name of a file in which future plots will be saved. Remember that each plot is about 25 disk pages. If you don't have enough room in your directory, you should connect to a scratch directory.

QUIT Exit the program.

29.3.2. Plotting Functions

Any valid function statement of up to 200 characters can be plotted. The expression can contain numbers, variables, operators, and functions. H2PLOT will not accept standard scientific notation, so you should use the "*10**n" notation. Up to 20 variables will be accepted; each must start with a letter and must be no longer than 5 alphanumeric characters. Only 2-dimensional plotting is allowed, although you can type an expression with several variables, then use each in turn on the x-axis while holding the others constant at a value you can specify. Due to a bug in the expression parser you may not use an expression such as

```
-t*sin(y)+qwert**-t
```

Instead you must write it as

```
(-1)*t*sin(y)+qwert**(-1*t)
```

Acceptable operators are: +, -, *, /, and ** . The available functions are: SIN, COS, ASIN, ACOS, ATAN, SINH, COSH, TANH, SQRT, ABS, EXP, LOG (common log), and LN (natural log). Trigonometric functions use radians.

Some of the more useful EXPRESSION mode commands are:

ADD Add the current data to the present plot. This command may be issued more than once to superimpose several sets of data.

CONSTANTS Enter a new set of constant values for the unspecified variables in the current expression. This applies only if you have entered more than one variable name.

DISPLAY Display the current function on the screen.

ENTER Enter a new expression. This command is automatically executed the first time you enter the EXPRESSION command mode.

NEW The next plot is to be a new one.

PRINT Send the current plot to the line printer or write it into a file if the PLOTFILE command was given previously.

QUIT Return to the H2PLOT top level.

SHOW Show the current function, range, and constants with their values.

RANGE Sets or changes the range of the x-axis. Type the word RANGE followed by the two numbers with no separating comma.

XVARIABLE Sets or changes the x-axis varible if you entered more than one variable in the expression.

29.3.3. Plotting Data Points

The DATA command mode allows you to read data from a file or to directly enter the points you wish to plot. If you have more than a few plots, it is best to put the data points in a file. Then if you happen to make a mistake, you don't have to start completely over. Up to three thousand points are accepted.

The DATA mode commands include:

ADD	Add the current data to the present plot. This command may be issued more than once to superimpose several sets of data.
CLOSE	Used to close the output file to which you may be writing your data points or to close an input file that contains more data than you want.
COLUMN	You may have several columns in your file. This command is used to select a particular column. The first column is always assumed to be the x-axis values, and the second is the default for the y-axis.
DISPLAY	Display on your screen a graph of the current data.
NEW	Start a new plot.
PRINT	Send the present plot to the line printer or write it into a file if a PLOTFILE command was given previously.
QUIT	Return to the EXPRESSION command mode.
READ	Read data in from a data file (extension DAT). The first line should contain a title. Several data sets, separated by blank lines, may be contained in the same file.
REARRANGE	Enter the data rearrangement command mode.
TYPE	Type out on the screen the current data points.
WRITE	Write out the current data points into a separate data file with a name you specify.

29.3.4. Rearranging Data Points

The REARRANGE command mode allows you to add or delete data points from the current plot. It also allows you to manipulate the current data in other ways using the X and Y commands. While in the rearrange mode, type a "?" after the X and Y commands to get the specific options. The TYPE command lists the current points on the terminal.

29.3.5. Curve Fitting

Your data can be fit by least-squares approximation to a number of functions including:

$\Sigma a_i X^i$	Simple polynomial
$\Sigma a_i X^{1/i}$	Inverse power series
$\Sigma a_i 1/X^i$	Inverse polynomial function
$\Sigma a_i \sin(X*i)$	Sine function
$\Sigma a_i \exp(X*i)$	Exponential function

The maximum degree of fit, or number of terms in the expansion, is ten. Curves are plotted as five hundred evenly spaced points from the function. This section has no explict commands but rather leads you through the fitting process each time you enter.

29.3.6. Changing Parameters

The PARAMETER command mode allows you to change the default plotting parameters such as size, bounds, grid, and titles. The default values for these are as follows:

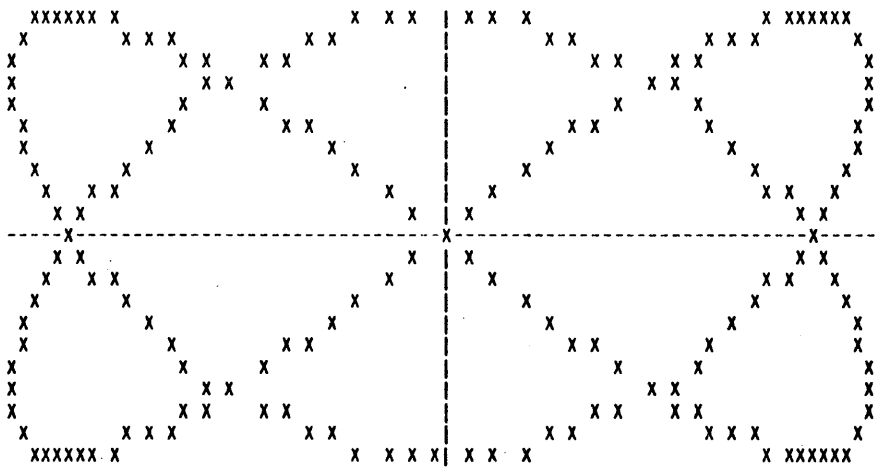
X-axis title	Previous title or variable from function plotting.
Y-axis title	Previous title or expression from function plotting.
Main title	Previous title or title from data set.
Bounds	Determined internally.
Plot size	24 by 24 cm (including all titles).
Grid	Suppressed.

To find out about the specific commands, enter the PARAMETERS command mode and type a question mark to list the commands. You can then type "HELP" followed by a command name to get an explanation of that particular command. If you like H2PLOT's default parameters, you don't need to use this command mode. If you do want to change the defaults do so *before* you enter the DATA or EXPRESSION command modes. The most common use of this mode is to change the titles on the plot.

29.3.7. Some H2PLOT Examples

In the following example we read data from a file and plot it. The axis and other parameters are all chosen automatically. We do request, however, that the plot be put into a file so it can be printed later. We also request that the data points be connected. After the file is read, the points are automatically shown on the screen.

```
@h2plot
      =>   Welcome to H2PLOT   <=

If you do not know what to do , try typing ? or HELP.

H2PLOT>   plotfile

File:  output
H2PLOT>   parameters
P-PARM>   character
Enter the character you want plotted:
(↑A connects points, ↑B gives sequential numbers, ↑D gives dots)  ↑A
P-PARM>   data
P-DATA>   read
Enter data file name [<CR>=TTY:=data entry from terminal; ? for help]

File name = lissaj.dat
```

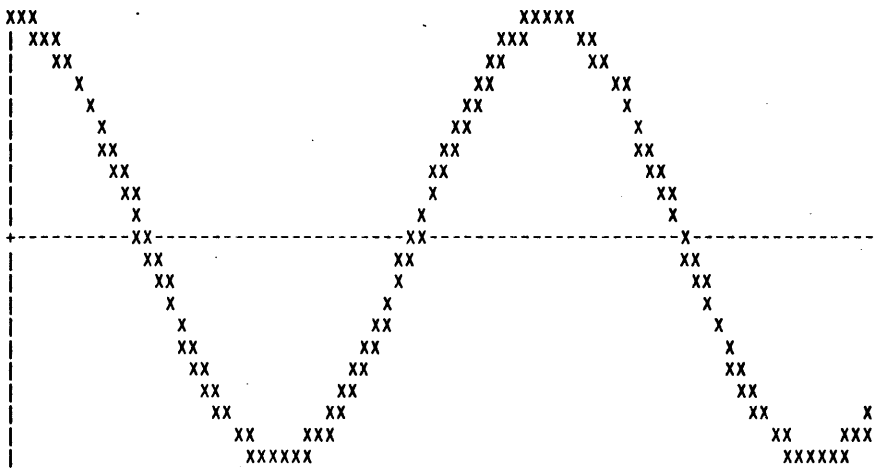


```
P-DATA>   show

 199 data points
Current plot number:   0
Next ADD creates a new plot.
Reading from column 1+ 1 for Y values.
Not listing data on lineprinter.
Auto Display
P-DATA>   add
P-DATA>   print

Sending current plot to lineprinter or file.
P-DATA>   quit
H2PLOT>   quit

STOP

END OF EXECUTION
CPU TIME: 3.88  ELAPSED TIME: 62.36
EXIT
@
```

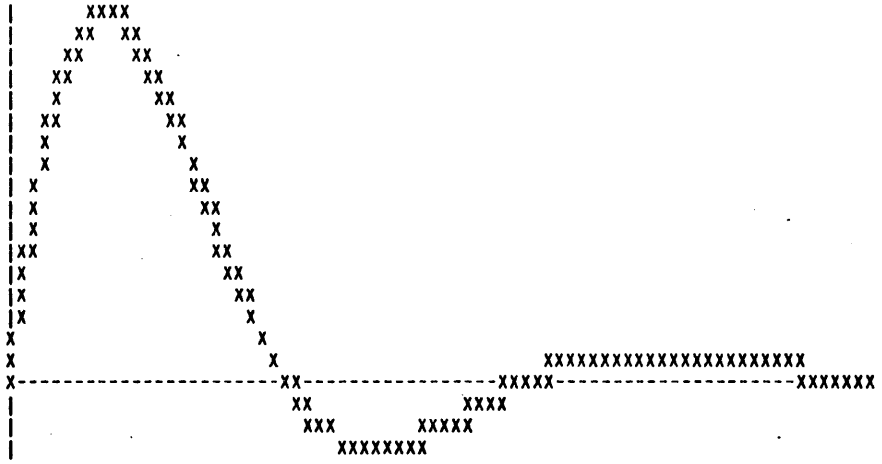
In the next example we use the EXPRESSION command mode to superimpose two sine curves on the same plot. Note that when you display the second time, you only see on the terminal the most recent effort. The other curves remain until you start over by typing the NEW command. If you do not want to add an expression to the plot you are constructing, do not say ADD as in the example. We also use the PARAMETERS section to put on a main title.

```
@h2plot
      =>   Welcome to H2PLOT   <=

If you do not know what to do , try typing ? or HELP.

H2PLOT>   parameters
P-PARM>   maintitle

This is the current main     title:

Enter the new main     title (return to blank;↑A for same)!
Sine Waves
P-PARM>   expression
The first thing you must do is enter the expression  you wish to plot.
Enter expression [Type: "?<CR>" for help]
5*sin(x+pi/2)


These variables were decoded:
   X
   PI

More than 1 variable was decoded. Use the
XVARIABLE command to specify which one you wish to vary
P-EXPR>   xvariable

Which parameter do you wish to vary?     x
P-EXPR>   constants


Enter the values of the other parameters:
PI        3.14159
P-EXPR>   range 0 10
P-EXPR>   show

The current expression is:
5*sin(x+pi/2)
The current variables are:
   X              0.0000000
   PI             3.1415900

The x-axis variable is:     X
The range of the x-axis is       0.0000000   to      10.0000000
```

```
P-EXPR>   display
```



```
P-EXPR>   add
P-EXPR>   enter

Enter expression [Type: "?<CR>" for help]
5*exp(-x/2)*sin(x)

SYNTAX FRROR:  missing operand...try again
Enter expression [Type: "?<CR>" for help]
5*exp((-1)*x/2)*sin(x)

These variables were decoded:
   X

P-EXPR>   show

The current expression is:
5*exp((-1)*x/2)*sin(x)
The current variables are:
   X             10.0000000

The x-axis variable is:     X
The range of the x-axis is       0.0000000   to      10.0000000

P-EXPR>   display
```



```
P-EXPR>   add
P-EXPR>   print

Sending current plot to lineprinter or file
P-EXPR>   quit
H2PLOT>   quit

STOP

END OF EXECUTION
CPU TIME: 5.63  ELAPSED TIME: 2:48.54
EXIT
@
```

29.4. The LOTPLT Package

LOTPLT is a set of FORTRAN routines which can be called from FORTRAN, SAIL, and Pascal programs and which allow one to write dot-plot files for the Printronix line printers. LOTPLT forms the basis of the H2PLOT program. This package is similar to OMNIGRAPH, but has many fewer features. People looking for a plotting subroutine package to learn should use OMNIGRAPH instead. Documentation on LOTPLT is in the file `DOC:PLOTTING.DOC`. The documentation provides examples of the use of LOTPLT in FORTRAN, Pascal, and SAIL programs. It is not possible to use LOTPLT with Passgo.

30. Statistical Computing on the DEC-20

30.1. Introduction

On the Stanford DEC-20's there is a large variety of statistical packages and some statistical/mathematical subroutines, usually most in FORTRAN. This section describes the statistical and mathematical software currently available and its use. A brief comparison of the statistical packages is also given.

Beware that not every package described herein may be available on your system. Since LOTS and the GSB Computer Facilities have a large number of social scientists among their users, much of the statistical software described in this chapter will be found at those two facilities. The numerical analysis software is found on nearly every Stanford DEC-20. If you can't find the program you're looking for, contact the system staff for assistance.

30.1.1. Statistical Packages

The statistical packages currently supported are SPSS, SCSS, MINITAB, TSP, and BMDP-81. MATLAB, an interactive matrix manipulation package oriented toward numerical analysis is privately supported at the GSB as are several other programs, to a greater or lesser degree, e.g., SIND (related to MDSCAL, for multidimensional scaling) and HICLUS (a hierarchical clustering program) to name but a few. Your well-documented statistical/mathematical software is welcome, assuming it is of general interest and currently in running order on a DECSYSTEM-20. The Graduate School of Business has several other packages, mostly aimed at financial modelling, e.g., EMPIRE, IFPS, LISREL, LINDO, LINMAP, and PEC. Contact the LOTS or GSB staff if you need to use these programs.

30.1.2. Subroutine Libraries

The Stanford DEC-20's provide the IMSL, NAG, and EDA statistical/mathematical FORTRAN subroutine libraries, a collection of routines callable from your FORTRAN, SAIL, or PASCAL programs. The IMSL library is generally regarded as being quite complete with respect to scope, well-tested and numerically sound. The NAG library has the usual collection of numerical utility subroutines, and some more recent numerical analysis algorithms as well. The EDA (Exploratory Data Analysis) library is a small subroutine library to do Tukey-type data analysis, as documented in Velleman and Hoaglin's "The ABC's of EDA". The libraries are implemented in single-precision and the subroutines are available as compiled library modules.

LOTS also undertakes to maintain the NALIB (Numerical Analysis LIBrary) and FORLIB as collections of user-supplied, documented and generally working subroutines. Most of these subroutines are written in FORTRAN, although some are written in PASCAL and a few in SAIL and MACRO. The scope of these libraries is limited to what past LOTS users have provided and maintained, but is, in general, not inadequate. Again, the addition of your well-documented statistical/mathematical subroutine or function is welcome, assuming it is of general interest and currently in working order on a DECSYSTEM-20.

30.2. Comparing the Statistical Packages

The statistical packages are all either FORTRAN programs or a collection of FORTRAN programs which reside, either directly or indirectly, on SYS:. In general, access to any package is obtained exactly as with any other system program, i.e. by typing its name to the EXEC's "@" prompt. The program then responds with its favorite prompt and you proceed. The packages do differ greatly, however, in their level of interactiveness and in their methods and the extent to which they handle I/O functions (reading and writing files).

A brief description and comparison of each package, including SPSS, SCSS, MINITAB, TSP, BMDP-81, and MATLAB will be given below. This section will be followed by a section for each package, providing a list of currently available documentation, an introduction to the use of the package, an introduction to package file I/O, and a sample session,

where appropriate.

30.2.1. Getting Help and Documentation

On-line help and documentation for each package may be obtained by searching for documentation files whose names match the names of the packages. For example, to locate help and documentation, respectively, for MINITAB, try

```
@vdirectory hlp:minitab.*
@vdirectory doc:minitab.*
```

to see the names of the available files and their sizes. The TYPE and PRINT commands may then be given to type the files at your terminal for smaller files, or printed on the lineprinter for larger files or for a permanent copy. Some of the package documentation files are quite large, and it is suggested that you avoid printing them and use the copies available at CERAS and Terman if possible.

30.2.2. Interactive Packages: MINITAB, SCSS, and MATLAB

- MINITAB: By far the most popular package at LOTS in terms of number of user sessions, MINITAB is an easy to use and rather complete package for many types of statistical analysis. Besides the usual descriptive and regression statistics, MINITAB offers ARIMA modelling and limited plotting.

  Advantages: MINITAB commands are short and sensible; the analyses are varied and quite complete; this is probably the easiest statistical package at LOTS to use. It is an excellent choice for smaller datasets.

  Disadvantages: MINITAB work spaces are not large; the package gets cumbersome with many variables or observations.

- SCSS: Really, Son of SPSS, SCSS was created to be the interactive counterpart to the larger non-interactive package. However, neither the command syntax nor keywords are very similar to SPSS. The package is well-documented internally and many SPSS users seem to prefer SCSS for interactive analysis.

  Advantages: The best feature of this package is its ability to read files written by SPSS using the SAVE SCSS procedure. In this way, larger data files may be subset for thorough analysis, interactively, a good feature for class use; output formats are nice; some graphic capabilitiess are available.

  Disadvantages: SCSS is limited in the scope of its statistical procedures; users seem to have difficulty creating and saving SCSS workspaces (called "masterfiles"); error documentation is not particularly good; package seems still fairly buggy.

- MATLAB: Not exactly a "statistical package" in the same sense as MINITAB and SCSS, MATLAB is really an interactive matrix manipulation package which has a variety of statistically oriented functions and applications. This package offers interactive help within the package and likely the best algorithms of any package in terms of state-of-the-art numerical analysis as well as the ability to incorporate user-written functions.

  Advantages: If it can be done to a numeric matrix, MATLAB can do it, and likely in a painless, elegant and numerically superior way; MATLAB takes conditional transformations (IF, WHILE, etc.), and can operate on submatrices and concatenate matrices easily; implementation of any statistical algorithm is really quite painless; workspace will hold 25,000 double-precision elements.

  Disadvantages: Requires a reasonably good working command of FORTRAN to understand the subscripting sytnax; the MATLAB command parser is relatively unsophisticated; file I/O is

adequate, but its effective use takes some practice and a *thorough* reading of the documentation.

30.2.3. Non-Interactive "Batch" Packages: SPSS, BMDP-81, and TSP

- SPSS: SPSS was the first "user oriented" statistical package, and for this reason enjoys a fair proportion of the canned software market. All of the SPSS documentation is unusually good both in terms of its readability and completeness. At present, SPSS is the only package available at LOTS and GSB to offer the features of the REPORT and RELIABILITY procedures; however, noting the two exceptions above, the package may, in general, be regarded as inadequate with regard to the scope of statistical analysis available, quite poor in the level of user specification possible (ability to change program defaults), and usually not the best terms of the numerical quality and selection of the statistical algorithms available. SPSS is particularly poor in situations involving ill-conditioned data.

  Advantages: SPSS has the most graphically pleasing output of any package; keywords are reasonably sensible; a variety of input formats are now readable with SPSS Version 9.1; easy data transformation and data subset capabilties; SCSS interface provides for easy interactive analysis of small data subsets; error documentation is the best of any available package. SPSS has recently expanded its nonparametric statistics section.

  Disadvantages: Limited statistical procedures; very limited user specification of statistical criteria; limited test statistics and visible intermediate calculations; limited plotting and graphics; least current of any package in terms of numerical analysis and robust algorithms.

- BMDP: Like SPSS, BMDP (series 81) programs are English language oriented and can perform a variety of statistical analyses with little user concern for algorithm details. In fact, this package was created to compete more effectively with SPSS in the novice user market. BMDP is structured much like its predecessor, BMD, in that it is not one program, but a collection of statistical subroutines, driven by a common command parser and I/O supervisor subroutine. Also, as with BMD, the algorithms are quite fast and efficient, although the high-level control language comes with a bit more overhead in this regard. BMDP has done an excellent job in updating the algorithms used for the various statistical analysis, providing, in general, the latest published numerical and statistical options and improvements for a given statistical technique. The BMDP package offers a very complete statistical library as well as a variety of plotting options.

  Advantages: BMDP control language is English-language oriented to the point of being constructed in sentences and paragraphs; algorithms are excellent; usually a variety of user-specified options with respect to plotting, test statistics and visibility of intermediate calculations; usually a choice of several optimization/calculation criteria as well as optional tolerance and test statistic specifications. This package is one of the best in terms of reliability and validity checking; data transformation with BMDP-81 is fairly flexible and sophisticated.

  Disadvantages: The BMDP Manual assumes a good bit of user statistical sophistication, devoting much space to the explanation of particular algorithms rather than package use; error documentation is inadequate, where present; control language is rather unforgiving of small syntax errors; worst feature of this package (as currently implemented on the DECSYSTEM-20) is its general inability to ***write out*** raw data files after transformation/augmentation by BMDP.

- TSP: The Time Series Processor is not a general-purpose statistical package in the sense of BMDP and SPSS; it is, rather, a collection of regression algorithms designed specifically for the analysis of time-series data. It is the only package with the capability of analyzing multiple-stage least-squares lagged models under a variety of assumptions and constraints. TSP is of BMD vintage, and the control language more closely resembles that of BMD than the niceties of SPSS or BMDP. TSP seems impervious to changes (improvements) in numerical computation as well as statistical innovation in the field of regression diagnostics, robust regression and

numerical stability of algorithms. TSP is also the only package which requires data in column-major order (as opposed to row-major) which makes either exclusive use of TSP or knowlege of FORTRAN an unpleasant necessity for many users.

Advantages: TSP offers the most general collection of regression algorithms for a wide variety of regression models; algorithms used are quite fast and efficient; package is generally well-tailored to econometric modelling and forecasting, although it is somewhat weak on the latter, especially with respect to nonlinear models; by far the most complete time-series package available at Stanford.

Disadvantages: TSP data files are incompatible with the rest of the statistical software at Stanford; algorithms are quite out-of-date, as mentioned above; command language is extremely primitive and unforgiving; package is quite given to user error; error diagnostics are inadequate; file I/O is inflexible and primitive.

30.3. Using BMDP

BMDP-81 programs are available at LOTS and GSB. All of the programs described in the new BMDP (1981) manuals are available and stored on the device `BMD:`. A brief guide to the BMDP's is available by typing:

```
@print doc:bmdp.doc
```

30.3.1. Documentation

Complete descriptions of the individual BMDP programs can be found in the following manuals:

- *BMDP Biomedical Computer Programs*, Health Sciences Computing Facility, Department of Biomathematics, School of Medicine, University of California Press, Berkeley, Los Angeles, and London (1979).
- Brown, M. B, et al., *BMDP-81, BMDP Statistical Software*, University of California Press, (1981). The primary BMDP-81 Manual.
- Hill, Mary Ann, *BMDP User's Digest -- A Condensed Guide to the BMDP Computer Programs*, BMDP Statistical Software, Department of Biomathematics, University of California, Los Angeles, (1980). A small digest of the programs.
- Forsythe, Alan B., *BMDP Reference Card*, BMDP Statistical Software, Biomathematics, UCLA, (1980)
- *BMDP Statistical Software, Communications*, Health Sciences Computing Facility. Current information about BMDP developments.

These documents may be purchased from most academic book sellers or directly from: Health Sciences Computing Facility, Center for Health Sciences, University of California, Los Angeles, CA 90024. The manuals are available for reference at each of the CERAS reference tables. One may also purchase the manuals at the Stanford Book Store or at I.T.S. Document Sales at Forsythe Hall.

30.3.2. Creating your Program

The first step in running an BMDP program is to place the source of your program in a file at LOTS. To do so, you would normally use the EDIT program (via the system commands `CREATE` and `EDIT`). This source file should have the file type "`.BMD`" to indicate that it contains an BMDP program.

Consider the following piece of a sample terminal session (commentary is in italics):

```
@terminal raise                                    set uppercase
@CREATE MYFILE.BMD                                 writing a new BMDP program
Input: MYFILE.BMD.1
00100   /PROBLEM TITLE IS 'BMDP RUN'.
00200   .
        ...
00500   .
00600   /END
00700   $ <--- Type the ESC key here, not "$".
*EU                                                Be sure to save the commands without line numbers.
[SURVEY.BMD.1]
@
```

Notes:

- Your program must be typed into the machine in upper case. The command `@TERMINAL RAISE` (abbreviated "`@TE RA`") causes the terminal to echo in upper case. To return to upper and lower case, give the command `@TERMINAL NO RAISE` (abbreviated "`@TE NO RA`"). BMDP programs *will not recognize* lower case letters.

- You must save *all* programs and input data files *without* any line numbers. To remove line numbers from the files in EDIT, save your program unnumbered (EU to the * prompt). Extraneous statement numbers will cause errors when running BMDP programs. A frequent and insidious bug occurs when line numbers are left in data files as the format statement will usually be written by the user so as to specify the line numbers as data.

30.3.3. How BMDP Finds Files

FORTRAN units referenced for data input and output to disk, and for writing SAVE files refer to files of the form `FORxx.DAT`, where `xx` is the unit number referenced, e.g., `FOR23.DAT`. Available unit numbers are 20-24 and 30-99. You may also specify the input file names in the `FILE` sentence of the `/INPUT` paragraph; the form is `FILE='NAME.EXT'`. Here the file name and file extension are specified and enclosed in single quotes. Note that the file name cannot have more than six characters and the extension not more than three. You may similarly specify an output file in the `/SAVE` paragraph.

In addition, the program you are using may create and use scratch units on disk. These unit numbers are given with each program in the appendix to the BMDP manual. Do *not* use these numbers as your data input or output units or redefine them. These scratch areas (files) should be scratched at the conclusion of your program. To do this, give the command `DIRECTORY` to the "`@`" prompt and look for `FORxx.DAT` or `FORxx.TMP` data sets where `xx` is the scratch unit or units listed for that program. These files may be deleted through the `DELETE` command.

The output files from the BMDP programs can be rather large which may create problems of disk storage (if you go over your working storage allocation). The program may stop running and the computer will suggest an `EXPUNGE` command followed by a `CONTINUE`. If this does not work, issue a `RESET` command and request more working storage from the computer facility staff if you do not wish to output directly to the printer.

30.3.4. Running a BMDP Program

Give the EXEC command `@BMD:BMDPxx`, where `xx` refers to the particular BMDP program you wish to run (e.g. BMDP1D). The computer will respond with an asterisk ("*"), which is a prompt for the input and output file names for the run, in the form "<output file>=<input file>" where <input file> is the name of the file which contains your BMDP commands, and <output file> is either a file name, if you want your output in a disk file, `TTY:` if you want your output directly on your terminal, or `LPT:` if you want output sent directly to the line printer. It is recommended that one avoid the latter two options and simply have BMDP create a disk file and then read/print the output file if desired.

Output and input are both standard TOPS-10 file specifications of the form

```
*dev:file.ext[p,pn]=dev:file.ext[p,pn]
```

where "dev" is any device (if not specified, it defaults to DSK:). "File" is any legal six-character filename. "Ext" is any legal three-character extension (which if unspecified defaults to .DAT). "[p,pn]" defaults to the user's project, programmer number. For example,

```
@BMD:BMDP1D
*SURVEY.LST=SURVEY.BMD
```

where SURVEY.BMD is the BMDP command file which will be given to the program, BMDP1D. BMDP1D will, in turn, write an output file called SURVEY.LST. The extension ".LST" is special and is usually reserved for output files as it will delete itself after it has been printed on the lineprinter. To print a BMDP output file, type, for example:

```
@print survey.lst/file:fortran
```

the /FILE switch is necessary as the output contains FORTRAN carriage control characters.

30.4. Using MATLAB

MATLAB is an interactive computer program that serves as a convenient "laboratory" for computations involving matrices. It provides easy access to matrix software developed by the LINPACK and EISPACK projects. The capabilities range from standard tasks such as solving simultaneous linear equations and inverting matrices, through symmetric and nonsymmetric eigenvalue problems, to fairly sophisticated matrix tools such as the singular value decomposition. MATLAB was produced by the University of New Mexico, Department of Computer Science, under the direction of Cleve Moler.

30.4.1. Documentation

Short MATLAB help files are available by typing:

```
@type hlp:matlab.hlp
```

An extensive documentation file is also available. If you are going to use MATLAB seriously, you should probably print yourself a copy:

```
@print doc:matlab.doc
```

Many references for MATLAB algorithms can be found in *Computer Methods for Mathematical Computation* by Malcom, Moler and Forsythe (Prentice-Hall, 1977).

30.4.2. How MATLAB Reads and Writes Files

The default unit for both input and output is the user's terminal. However, MATLAB has the facility for both reading and writing data files to the user's disk directory.

Entire worksheets may be saved using the SAVE and LOAD commands. Alternately, you may specify particular arrays be saved, if you wish. Stored constants and matrices are all stored on disk. The commands are:

```
SAVE      SAVE('file.ext')      to save all current variables
SAVE(A)   SAVE('file.ext',A)    to save only array A

LOAD      LOAD('file.ext')      to read in all saved variables
LOAD(A)   LOAD('file.ext',A)    to read in only array A
```

File names are enclosed in single quotes and may specify any legal file name, including directory name, logical device, and/or PPN if the file is not in the connected directory. The entire specification must be 30 characters or less, and the file name itself must be no more than 4 characters, and terminate with an acceptable extension of 3 characters or less.

There are essentially two ways to get data in and out of MATLAB. For lack of better terminology, there is the easy way and the hard way. The easiest way to get data into MATLAB is to enclose the datafile in <>'s, prefacing the leading "<" with "X=", where X is the desired variable name. Then simply give the EXEC command in MATLAB, indicating the

name of the modified data file. MATLAB will perform the indicated operations of the EXEC file, namely, reading X. Note that the columns of X must be separated by spaces. MATLAB does, however, provide the PRINT ('file.ext') facility for writing data out of MATLAB in a format of the user's choice.

The other way to get MATLAB to read your data file is to rewrite the file into "MATLAB" format, i.e., format your data as indicated below; MATLAB is a little different from the other packages in that the user may create the MATLAB file for MATLAB. There are examples of a FORTRAN program to read/write files in the MATLAB help and documentation files.

30.4.3. Using MATLAB

For updates on MATLAB and user news, see `DOC:MATLAB.NEWS`.

- MATLAB reads (and prints) a command line, looks up the command in its dictionary (of about 50 names), checks for errors, carries out the command, and then goes on to the next command. To use MATLAB, type

```
@matlab
```

- We use the word "argument" to mean either a number (e.g. 5.31 or 23), a stored constant (ALPHA, PLEVEL), a vector (X1, y, BHAT), or a matrix (M1, XPX, XPXI). Some commands want an argument; some don't. For information on any particular MATLAB keyword or function, type

```
<>
help THING<CR>                                   NOTE: CR = carriage return
```

where THING is the desired MATLAB keyword or function. Typing

```
<>
help<CR>
```

will get you the MATLAB help file, which you may print using the command

```
@print hlp:matlab.hlp                            Gets you the short helpfile.
```

or,

```
@print doc:matlab.doc                            Gets you the more extensive file.
```

- Usually each MATLAB command will be on a separate line. Exceptions to this will occur with the use of FOR, IF, etc.

- In some commands, you can use either numbers, stored constants, columns, or matrices as arguments. Example:

```
ABS(x)                          FOR i=1:N                          EXEC('file.ext')
```

- Two or more slashes ("//") indicate the logical end of the line. You may place comments following the // if you wish. This feature is especially helpful when using EXEC files.

- Any MATLAB command may be continued onto the next line by ending the line with two periods (". .") at the end of the line. This applies, also, to matrix element entry.

- Some commands produce a matrix (vector, scalar) answer. If you do not explicitly name the new result, that array will be stored with the name ANS. If you do not rename ANS (e.g., XPXI=ANS), the next time you write into ANS implicitly, the old ANS will be lost forever!

- The work area will store 25000 double-precision array elements. This is enough space for several large matrices and assumes judicious use of the CLEAR command.

- Matrices may be entered with rows separated by ';', e.g.,

```
A=<1 2 3;4 5 6;7 8 9>
```

would result in a 3 by 3 matrix with the elements listed above. Alternately, and for large matrices, rows may be delineated by a carriage return, e.g.,

```
A=<1 2 3
   4 5 6
   7 8 9>
```

- Commas cannot appear within a number, e.g., write 1241, not 1,241 for the number one thousand two hundred and forty-one.

- Data may be read into MATLAB via the LOAD command. This is implemented somewhat differently than most of the packages in that you must create the MATLAB 'save' file for MATLAB. For more information, see <>help LOAD in MATLAB, or, better yet, see the section on How MATLAB Reads and Writes Files.

- For even moderately extensive use of MATLAB, see the file in DOC:MATLAB. Dr. Moler and associates have demonstrated many nifty features and written up several very helpful examples in this document.

30.4.4. A Sample MATLAB Session

```
@matlab

     < M A T L A B >
   Version of 10/31/80
HELP is available
<>
load('matx.dat')                          // get data from file on disk
<>
who

Your current variables are...
X     EPS   FLOP  EYE   RAND
using      20 out of    5005 elements.

<>

X                                         // print X

X     =

   34.0000    6.4000    2.1000
   42.0000    6.9000    2.5000
   33.0000    5.8000    2.5000
   41.0000    6.5000    3.1000
   37.0000    5.6000    2.4000

<>
A=X(:,2:3)                                // grab last two columns for indep. vars.

A     =

    6.4000    2.1000
    6.9000    2.5000
    5.8000    2.5000
    6.5000    3.1000
    5.6000    2.4000
<>
```

```
y=X(:,1)                                    // first column is response vector

Y     =

   34.
   42.
   33.
   41.
   37.
<>
APAI=inv(A' * A)                            // get inv. of cross-products matrix

APAI  =

    0.2812   -0.6863
   -0.6863    1.7060
<>

Beta=APAI * A' * y                          // parameter estimates..

BETA  =

    3.8644
    5.2777
<>
SSE=y' * y-Beta' * A' * y                   // Error sums of squares

SSE   =

   22.1629
<>

DFE=2                                       // d.f. error = n-k-1 = 2

DFE   =

    2.
<>

MSE = SSE/DFE                               // estimate of sigma **2

MSE   =

   11.0815
<>

Cov=MSE * APAI                              // covariance of estimates

COV   =

    3.1164   -7.6058
   -7.6058   18.9053
<>
DCov=diag(COV)                              // get diagonal of covariance matrix

DCOV  =

    3.1164
   18.9053
<>

STDE = SQRT(DCov)                           // std. dev. of estimates

STDE  =

    1.7653
    4.3480
<>
for i=1:2,t(i)=(1/STDE(i))*BETA(i)          // t-statistic for Betas

TTST  =

    2.1891
    1.2138
<>
```

```
Beta

BETA  =

    3.8644
    5.2777
<>
exit                                          // end of OLS example

total flops       133

ADIOS
@
```

30.5. Using MINITAB

MINITAB is an easy to use statistical package designed for people with little or no computer background. MINITAB can calculate means, covariances, chi square statistics, linear regression coefficients, ARIMA models and a host of other useful things.

MINITAB is written in FORTRAN and is maintained by the Department of Statistics at Penn State University.

30.5.1. Documentation

The *Minitab II Reference Manual* is available from the Stanford Bookstore or I.T.S. Document Sales. Copies are kept at CERAS for reference. Perhaps, however, the MINITAB user's guide is more useful. These may be purchased at the Bookstore or at the GSB Computer Facility. The user's guide contains a general discussion of the features of MINITAB and a description of commands. If all you need is an annotated list of commands type:

```
@print doc:minitab.dat/file:fortran
```

Please save this listing for your future reference if you print it.

Another reference is the *Minitab Student Handbook*, by Ryan, Joiner and Ryan, Duxbury Press, which is available through the bookstore. This handbook contains most of the information in the reference manual as well as numerous examples of basic statistical operations using MINITAB.

As minor changes are made in MINITAB, bugs are found, etc., they will be documented in the file `DOC:MINITAB.NWS`.

30.5.2. Using the MINITAB Program

Before running MINITAB you must make your terminal type upper case only. This is done with the command `@TERMINAL RAISE` (abbreviated `@TE RA`) to the "@" prompt. When you are done using MINITAB you may return to normal terminal operation by giving the command `@TERMINAL NO RAISE` (abbreviated `@TE NO RA`). After you have "raised your terminal" you may type `MINITAB` to the EXEC to begin.

```
@terminal raise
@MINITAB

MINITAB will then respond with

***MINITAB II***
--
```

You are now running MINITAB and may use MINITAB commands. If output from a single MINITAB command fills the screen, the terminal will beep and stop typing; you may view the additional output by typing `CTRL/Q`. MINITAB uses the double slash ("--") as a prompt. After each command or data entry hit RETURN; MINITAB will then receive what you have typed in and answer with a -- (or possibly an error message, if it didn't like what you gave it).

30.5.3. Basic Rules of MINITAB

For the rest of our description of MINITAB the following notation will be in effect.

K	denotes a constant (such as 8.3 or 21) or a stored constant
C	denotes a column number, which must be entered with the format Ck, where k is an integer. An example is C2. By 'Column' we mean a column of numbers.
E	denotes an argument which can be either a constant or a column number.
[ ]	denotes an optional argument

The basis for all MINITAB commands is a command word followed by an argument or arguments. Arguments are either constants or column numbers. For instance, the command

```
PRINT C5
```

will cause MINITAB to print the contents of column five onto your screen. The command word must be the first part of the command. The command word and arguments may be separated by any useful English which you would like to include (Be careful if you try to put numbers between the command word or it's arguments). To see the contents of columns 5 and 6, we could have said

```
PRINT OUT THE CONTENTS OF C5 AND C6
```

A side effect of this feature is that constants may not contain any commas. Here are some rules:

- There may be only one command on each line
- If a command has both constants and columns as arguments, the columns must be given first. The command

  ```
  ADD C1 TO K1, PUT THE RESULT IN C2
  ```

  is OK, but

  ```
  ADD K1 TO C1, PUT THE RESULT IN C2
  ```

  is not.
- Only the first four letters of a command word are used by MINITAB; for instance, PRINT C1 can be entered as PRIN C1.
- The maximum length of a command line is 80 characters. To enter a command that is longer than 80 characters (which is the screen display width on most terminals) an asterisk must appear in the eightieth character position followed by a carriage return. You may continue the command on the next line.
- A list of consecutive column numbers may be indicated by using a dash, as in:

  ```
  READ C1-C3
  ```

  in place of

  ```
  READ C1,C2,C3
  ```

- If NOTE is the command word MINITAB will ignore what follows. This is the proper command to use for comments.
- If STOP is the command word MINITAB will halt and return you to the EXEC. You will no longer be "in MINITAB". This is the proper method for leaving MINITAB.

30.5.4. Some Sample Command Words

This is by no means a complete list of the command words available in MINITAB. The rest may be found by consulting the documentation mentioned above. The command words are given, followed by their arguments. On the next line is an explanation, followed in some cases by examples of use of the command word.

READ ['filename'],C,...,C

Read from the file named 'filename' the members of C,...,C. If the filename is not specified, MINITAB assumes that the input is from the terminal. If the filename is specified the file must be in your directory or else the directory it is in must be specified. The ' delimiters are necessary to distinguish a file name from other words in the command.The filename may not exceed ten characters in length. The input file may not contain edit line numbers. When you have finished creating or editing an input file you should leave with the EU Edit command.

```
READ FROM '<S.STATISTICS>TRY.DAT' THE MEMBERS OF COLUMN C1
```

This will read the entries of column C1 from the file named TRY.DAT in the directory named <S.STATISTICS>. Again, the filename may not exceed ten characters in length.

```
READ C1,C2,AND C3
```

This will read the entries for the three columns from the screen because no file was specified. The next thing you type must be the input to columns 1,2,3 separated by at least one space. Each row must be entered separately, followed by a carriage return. Input may be terminated by entering the command word END.

WRITe ['filename'],C,...,C

Write to file 'filename' the values in columns C,...,C.

```
WRITE TO 'SAMP.DAT' C1-C5
```

This will write the contents of C1 through C5 into the file named SAMP.DAT which will be created (if it doesn't already exist) in your directory.

```
WRITE '<F.FRIEND>FIL.DAT' C1
```

This won't work unless you have write access to the directory <F.FRIEND>. If you do happen to have write access to this directory (probably because you have connected to it) you will output to the file FIL.DAT in the directory <F.FRIEND>.

PRINt C,...,C Outputs the columns on your screen.

ADD E,...,E,C Add E to E and put the sum in column C.

```
ADD C1 TO 5 AND PUT THE SUM INTO C5
```

This will add 5 to each member of column 1 and put the new column into C5.

```
ADD C1 TO C2 AND PUT IN C3
```

This will add column one to column two member by member. If the two columns are not equal in length the result will be the same length as the shorter one, and a message will be printed on the terminal.

AVERage C,[C] Find the arithmetic mean of column C and print it on the screen, optionally store the value in the second column.

```
AVERA THE VALUES IN COLUMN C3 AND PLACE THE VALUE IN C1
```

Output the average and also store it in C1.

```
AVER C2
```

Output the average of the values in C2.

These instructions are more or less typical. For more detail, consult the MINITAB User's Guide.

30.5.5. Writing Files in MINITAB

MINITAB can save sequences of commands for the purpose of using them in another MINITAB session or to be repeated a number of times. MINITAB can also save its *worksheet* (all the columns and constants you have given it).

Saving Commands:

Commands may be stored in a disk file by using the STORE command word with an argument nn. nn must be between 20 and 24 or between 30 and 63. If no nn is given, 23 is assumed. The number nn determines the name of the file in which the commands will be stored; the filename will be FORnn.DAT. The command

```
STORE ON UNIT 21
```

will cause the commands to be stored in the file FOR21.DAT in your directory. The commands given after the STORE command will be executed and stored until the command END is given. The commands will be executed K times when the command EXECUTE nn,K is given where nn is the same number given in the STORE command.

Saving the Worksheet:

The worksheet may be saved with the SAVE [nn] command, where nn must be between 20 and 24 or 30 and 63. The worksheet will be stored on disk in the file FORnn.DAT. To retrieve the worksheet during a subsequent run, give the command RETRIEVE nn. Don't try to save two different worksheets in the same file.

30.5.6. A Sample MINITAB Session

```
@create inptry.dat
Input: INPTRY.DAT.1
00100   1
00200   2
00300   3
00400   $   (escape key)
*eu

[INPTRY.DAT.1]

@terminal raise
@MINITAB

*** MINITAB II ***
--
READ C1

--
5

--
7

--
13

--
END
```

```
--
READ FROM 'INPTRY.DAT' COLUMN C2

COLUMN       C2
COUNT         3

          1.          2.          3.

--
AVER C2 AND PLACE THE RESULT IN C3

   AVERAGE =       2.0000
--
ADD C3 TO C1 INTO C4


--
WRITE TO 'OUTTRY.NUM' ,C4

THIS FILE WAS WRITTEN USING THE FORMAT

(F4. 0,F4. 0,F4. 0,F4. 0,F4. 0,F4. 0,F4. 0,F4. 0,F4. 0,F4. 0,F4. 0,F4. 0,F4. 0,F
4. 0,F4. 0,F4. 0,F4. 0,F4. 0,F4. 0,F4. 0)
--
STOP

*** MINITAB II *** STATISTICS DEPT * PENN STATE UNIV * RELEASE 79.0 *
STORAGE AVAILABLE 10000

STOP

END OF EXECUTION
CPU TIME: 1.50  ELAPSED TIME: 2:20.81
EXIT
@TERMINAL NO RAISE
@type outtry.num
  7.  9. 15.
```

30.6. Using IMSL, NAG, NALIB, and FORLIB Subroutines

Several separate subroutine libraries are available. The most complete of these is the IMSL library, Version 9.0, which consists of a substantial and well tested collection of subroutines and function subprograms in the areas of mathematics and statistics. The NAG subroutine library is somewhat less general, but more current in terms of the quality and choice of algorithms. In addition, user-written subroutines in mathematics and statistics are contained in the LOTS NALIB. Other FORTRAN utilities are in the FORLIB library, as well as a variety of other subprograms which are unrelated to FORTRAN.

The IMSL library contains routines for handling problems in the following areas:

- Analysis of variance
- Basic statistics
- Categorized data analysis
- Differential equations; quadrature; differentiation
- Eigensystem analysis
- Forecasting; econometrics; time series; transforms
- Generation and testing of random numbers
- Interpolation; approximation; smoothing
- Linear algebraic equations
- Mathematical and statistical special functions
- Nonparametric statistics
- Observation structure; multivariate statistics
- Regression analysis
- Sampling
- Utility functions

- Vector and matrix arithmetic
- Zeros and extrema; linear programming

The NAG library contains routines for handling problems in the following areas:

- Complex Arithmetic
- Polynomials, Roots of Transcendental Equations, Series Summation
- Quadrature
- Ordinary and Partial Differential Equations, Differentiation
- Integral Equations, Simultaneous Linear Equations
- Interpolation, Curve and Surface Fitting, Minima and Maxima
- Approximation of Special Functions
- Matrix Operations, Eigen Analysis, Determinants, Orthogonalization
- Simple Statistics, Correlation and Regression, Analysis of Variance
- Nonparametric Statistics
- Operations Research
- Random Number Generation
- Sorting

The EDA library contains routines for Exploratory Data Analysis *a la* Tukey:

- Stem and Leaf Displays
- Histograms
- Median Statistics
- Boxplots
- x-y Plots
- Smoothing and Local Line-Fitting
- Rootograms

The NALIB contains two sets of routines:

1. The old LOTS Numerical Analysis Library routines which handle single-precision reals.

2. The Kane routines, which include double-precision and complex analogs of some of the Numerical Analysis routines above, in addition to some specialized routines with appliciations to specific courses.

The FORLIB library contains user-donated FORTRAN-callable subroutines and programs.

30.6.1. Documentation

Help for the libraries is available by typing `HELP IMSL`, `HELP NAGLIB`, `HELP EDALIB`, `HELP NALIB`, or `HELP FORLIB`, respectively.

The IMSL library is documented in the *IMSL Library Reference Manual* (4 volumes); NAG is documented in *NAG Library Reference Manual* (6 volumes). Both manual sets are available on the bookshelves in CERAS and Terman, as well as at the GSB Computer Facility and Department of Computer Science, 3rd floor Numerical Analysis terminal room.

Routines from the Numerical Analysis Library are documented in files named `NAL:*.DOC` where "*" is replaced by the name of the routine you would like to read about. Documentation for some routines is available in binders in the CERAS lobby. All programs depend upon internal documentation for specifics concerning argument lists and types and subprograms called by the subroutine itself, if any. Some programs will have an external documentation file. In that case, `PROG.DOC` will reside on the `NAL:` along with `PROG.REL`.

Routines from the FORLIB library are documented in `HELP FORLIB`. Each section of the FORLIB has a subsection in `HLP:FORLIB.HLP`. See this file for details.

30.6.2. Calling a Library Subprogram

To include a subroutine from either the IMSL library, FORLIB, or NALIB with a main program of your own, you should give a command similar to one of the following:

```
@execute myprog.for,IMSLIB:/lib
@execute myprog.for,NAGLIB:/lib
@execute myprog.for,EDALIB:/lib
@execute myprog.for,NAL:NALIB/lib
@execute myprog.for,NAL:FORLIB/lib
```

Note that the name of the IMSL library is defined by the system as `IMSLIB:`. Similarly, the NAG library is `NAGLIB:`, and the EDA library is `EDALIB:`. However, the NALIB and FORLIB libraries appear in files `NAL:NALIB.REL` and `NAL:FORLIB.REL` respectively. The "`/LIB`" informs the linkage editor that the specified file is a library and instructs it to load only the part(s) of the library which your main program requests, thus saving you memory space.[37]

For those of you who wish to see the original source programs, please contact the LOTS staff for the location of the source directory that contains all of the programs which were used in building the libraries. You may make copies or listings as need be, but please remember that the originals are not necessary unless you need to either modify the subprogram or know the exact algorithm employed by the subprogram.

30.7. Not Using SAS

SAS is a large and complex package which is as much a database language as it is a complete and well-designed statistical package. Very regrettably, SAS is written in BAL (IBM Assembler) and PL/I in addition to FORTRAN, the former language (BAL) being IBM hardware-dependent and PL/I not available as yet for the DECSYSTEM-20. In addition, SAS file I/O is heavily tied to the IBM operating system which is in turn IBM hardware-dependent. Therefore, in addition to rewriting most of the code for the SAS command interpreter and some statistical procedures, all routines for file I/O, including internal workspace management would need to be rewritten. This is not likely to happen in the near future; this section serves only to answer an oft-asked question, "Why doesn't LOTS have SAS?".

30.8. Using SCSS

SCSS is a fully conversational system that provides the following statistical techniques:

- Univariate frequencies, histograms, and statistics.
- N-way crosstabulations with statistics.
- Breakdowns of means with statistics and means plots.
- Scatterplots and casewise plots.
- Product moment correlations and significances.
- Partial correlations.
- Stepwise regression and creation and display of residuals.
- Factor analysis and creation of factor scores.

The data to be analyzed can reside on an ASCII file, can be entered into SCSS at the terminal, or can be generated by the SAVE SCSS command in the SPSS batch system. Once entered into SCSS, the data are in the form of an SCSS masterfile and can be accessed in this form in subsequent sessions. Version 4.2 of SCSS is currently available at LOTS and GSB.

30.8.1. Documentation

- SCSS: *A Guide to the SCSS Conversational System*; Nie, Hull, Franklin, et al.; McGraw-Hill, 1980

---

[37] You can also build and maintain your own subroutine libraries by means of the MAKLIB program. Say `HELP MAKLIB` for details.

30.8.2. Using the SCSS Program

SCSS prompts for the information it requires. Details on the syntax of the anticipated response to any prompt may be elicited by typing /HELP. A more verbose explanation of what SCSS expects from you results from typing /EXPLAIN.

TOPS-20 input recognition may not be used; however, CTRL/U, CTRL/R, CTRL/W, etc., work as usual. The backspace and strikeover method of correcting input errors works. SCSS can be interrupted with one or two CTRL/C's.

30.8.3. How SCSS Finds Files

SCSS builds workfiles during the SCSS session. These workfiles may be named, and the workfile name is denoted by taking a "&" as the first character (for those of you familiar with IBM systems, the "&" denotes a temporary dataset). These workfiles may be saved as SCSS "masterfiles" and reread into SCSS at a later time, much the same as SPSS saves its systems files.

SPSS can create an SCSS masterfile using the SAVE SCSS procedure in SPSS. This card replaces the SAVE FILE card where the user wishes to save an SCSS masterfile rather than an SPSS system file. Both the masterfile and system file may not be saved in the same SPSS run.

SCSS has the ability to write the output to a disk file when the "SCRIPT ON" facility is invoked. As with the other interactive packages, SCSS can take data input directly from the terminal during the interactive session, saving those data, if requested, in a masterfile for later use.

30.8.4. A Sample SCSS Session

```
@SCSS
LOGFILE OK
EXPLAIN? (Enter YES or NO)
NO
STYLE OF PROMPTING?
NORMAL
WORKFILE NAME?
NONE
NEW WORKFILE NAME?
&VOTER
MASTERFILE NAME?
FILE56
PROCEDURE?
/SCRIPT ON, DEMO.TXT                    /* This file gets output
SCRIPT OK
PROCEDURE?
/PROCEDURE UNIVARIATE                   /* UNIVARIATE gives stats
VARIABLES LIST?
/VARIABLES PRESVOTE, SESINDEX, AGE, FBELIEF
CELLS?
DISPLAY MEAN, STDDEV, MEDIAN            /* Requested statistics

VARIABLE        N       MEAN    STD DEV         MEDIAN          LABEL

PRESVOTE      1380      2.304      1.430          2.000        PRES VOTE CHOICE
SESINDEX      1287       -.058      .983           -.118       OCC. STATUS
AGE           1490     45.166    16.318         43.000         RESPONDENTS AGE
FBELIEF       1388     18.999   182.929         31.000         FATHERS BELIEF

TOTAL N=1500
```

```
DISPLAY?
/TIME
  33.83 SECONDS CPU TIME
   4.01 MINUTES ELAPSED TIME
PROCEDURE?
/SCRIPT OFF        /* This is like turning PHOTO off
SCRIPT OK
PROCEDURE?
STOP
@
```

30.9. Using SPSS

SPSS (Statistical Package for the Social Sciences) is available in a version (SPSS-10 Release 9.0) corresponding to the IBM-SPSSM Release 9.

30.9.1. Documentation

This section contains hints on how to write, run, and debug an SPSS program. It is oriented towards users who already know SPSS, but who are not familiar with the version of SPSS on the DEC-20. DEC-20 SPSS is almost identical with versions of SPSS available at other installations. Hence, all the published SPSS documentation still applies. With a few exceptions noted here, there are essentially only procedural differences in its use on the DEC-20.

SPSS is documented in:

- Klecka, Nie, Hull. *SPSS Primer*, McGraw-Hill, 1975. An excellent introduction to SPSS and statistical computing in general, available at the Stanford Bookstore.
- Nie, Hull, Jenkins, Steinbrenner, Bent. *Statistical Package for the Social Sciences*. Second Edition, McGraw-Hill, 1975. The standard reference manual for SPSS, available at the Stanford Bookstore.
- Nie, et al. *SPSS Update, Version 7-9* Second Edition, McGraw-Hill, 1981. The updated SPSS Manual, containing REPORT and RELIABILITY documentation, as well as other updated information. Available with the SPSS Manual. You need this manual to use SPSS successfully.
- Nie, et. al. *SPSS Algorithms*, McGraw-Hill. .Contains a description of the statistical algorithms used by the various procedures in SPSS. Available from directly McGraw-Hill and occasionally from I.T.S. or the Stanford Bookstore.

SPSS documentation is available on-line as the file `DOC:SPSS.DOC`.

30.9.2. Creating your Program

The first step in running an SPSS program is to place the source of your program in a file. To do so, you would normally use the EDIT program (via the system commands `CREATE` and `EDIT`). This source file should have the file type "`.SPS`" to indicate that it contains an SPSS program.

Consider the following piece of a sample terminal session (commentary is in italics):

```
@terminal raise                    set uppercase
@CREATE MYTEST.SPS                 writing a new SPSS program
Input: MYTEST.SPS.1
```

```
00100   RUN NAME        EXAMPLE - CREATING AN SPSS SAVE-FILE
00200   VARIABLE LIST   V1 TO V13
00300   INPUT FORMAT    FIXED (3F1.0,10F2.0)
00400   INPUT MEDIUM    MYFILE.DAT
00500   N OF CASES      49
00600   ASSIGN BLANKS   -1
00700   MISSING VALUES  V1 TO V3(-1,9)/V4 TO V13(-1,99)
00800   VAR LABELS      V1,AGE/
00900                   V2,SEX/
01000                   V3,ETHNIC GRP/
01100                   V4,CPU FAMILIARITY
01200   VALUE LABELS    V1 (1)YOUNG (2)OLD/
01300                   V2 (0)FEMALE (1)MALE/
01400                   V3 (1)WHITE (2)BLACK (3)ORIENTAL
01500                      (4)AMERIND (5)NONWHITE OTHER
01600   CROSSTABS       TABLES=V2 BY V4
01700   STATISTICS      ALL
01800   READ INPUT DATA
01900   SAVE FILE       MYFILE.FIL
02000   FINISH
02100   $ <--- Type the ESC key here, not "$".
*EU                                   Be sure to save the commands without line numbers.

[MYTEST.SPS.1]
@
```

Several points should be noticed about this example. First, SPSS, unlike most of the programs on the DEC-20, requires that your programs be in upper case only, and that they be saved in files that do not contain EDIT line numbers. It is convenient to give the EXEC command `@TERMINAL RAISE` (abbreviated `@TE RA`) to instruct the system to simulate a typewriter shift-lock, so you will not have to constantly hold down the shift key. Second, always be sure to exit from EDIT with an `EU` command to save your program "unnumbered". Finally, SPSS cannot read anything in a line typed past column eighty. Be sure to use a carriage return at or before the eightieth character of each line in your SPSS program; if you type a very long line it will wrap around on the terminal screen, making it appear to occupy several lines. As you type your program using the EDIT program, EDIT will give you a line number prompt at the beginning of each line. Do not break a line (insert a carriage return) in the middle of any word, variable name, value specification, etc. Continue a line on or after column 16 on the next line.

Users familiar with IBM SPSS will note rather strange SPSS commands in lines 400 and 1900, which illustrate useful extensions to standard SPSS. The INPUT MEDIUM card in the example instructs SPSS to look for its data in the file whose name is `MYFILE.DAT` ("DAT" is a standard file type for raw data files on the DEC-20). This syntax is very convenient if you have a fairly large amount of data which you will not be editing frequently -- you can keep your data in a different file from your program, and edit only the one that needs to be changed. SPSS will save the file information including data in `MYFILE.FIL` (".FIL" is the standard extension for SPSS system files at LOTS). ***Do not use tabs in your data*** because they are read by the computer as one character even though they may appear on the terminal as several spaces. Use multiple spaces instead.

Note also line 600. The ASSIGN BLANKS card replaces the "BLANK" keyword on RECODE statements, and is used to instruct SPSS to treat blank fields in fixed-field input data specially. In the example above, blank fields are treated as if they contained the value -1 instead of the default value of 0, making ASSIGN BLANKS equivalent to:

```
RECODE      ALL (BLANK=-1)
```

30.9.3. How SPSS Finds Files

File specifications for SPSS consist of a name-part, a period, and a file type, plus optional fields to refer to files belonging to other users. SPSS requires that the name-part contain six characters or less, and that the file type be three characters or less. Thus, a typical file name might be `MYTEST.SPS`.

The name-part is your choice, but conventions govern the standard choice of the file type. Use the type `.SPS` for SPSS control card files only, `.FIL` or `.SYS` for SPSS system files, `.DAT` for raw data files, and `.LST` for output. Then a `DIRECTORY` command will show you exactly what you have.

The general syntax for file names is

```
dev:file.ext[p,pn]
```

If you need to refer to a file in some other user's directory, note that SPSS does not understand directory names in angle brackets, but uses the old style PPNs (project-programmer numbers) instead. To find out what PPN a given directory corresponds to, give the TRANSLATE command to the EXEC. Thus, to refer to a file belonging to J.JQJOHNSON you might say:

```
@translate <j.jqjohnson>
PS:<J.JQJOHNSON> (TS) PS:[4,224]
@spss
*lpt:=mytest.sps[4,224]
*^Z
@
```

Alternatively, you can define a logical name to refer to the directory you are interested in.

```
@define jq: <j.jqjohnson>
@spss
*lpt:=jq:mytest.sps
*^Z
@
```

30.9.4. Running an SPSS Program

Running an SPSS program consists of calling the SPSS system, then telling it where to send the output, where the input (source program) is coming from, and what special switches are needed to control the run. Most of the time, though, SPSS will make reasonable assumptions about the output file and necessary switches, so you need only specify the input file. In the following example the input comes from the file created above, and the listing output goes to the user's disk area.

```
@spss
*myfile
*^C
@
```

SPSS prompts with an asterisk for instructions for one run. After it has finished with that run, it again prompts with a star, but since we have no more runs to make at the moment we terminate the SPSS program with CTRL/Z or CTRL/C.

The most general form (where the input stands for the name of the setup file you created above) is the following:

```
OUTPUT=INPUT,SWITCHES
```

If your file has a type of ".SPS", it is unnecessary to specify the entire input filename (as above). Also, MYTEST.SPS will by default generate the output file MYFILE.LST, though you may choose to have SPSS create the output file with any other permissible file name.

You get three choices for the output destination: "TTY:" will send the output to your terminal; "LPT:" will send the output to the line printer; a file name will send output to that disk file. Thus, the command

```
*lpt:=mytest.sps
```

will send the output directly to the line printer, whereas

```
*output.lst=mytest.sps
```

will send the output to the disk file (in your account directory) named OUTPUT.LST.

Sending your output to a file is an excellent idea, since you can then

```
@type output.lst
```

to check if the run worked, and

```
@print output.lst/file:fortran
```

to send the output to the line printer after you have verified that the output is what you want. Note that listings generated with SPSS include carriage control characters in column 1, and hence should be printed with a /FILE:FORTRAN switch; do not use this switch for SPSS programs or data files.

You can give various switches to specify special requirements of a run, analogous to parameters that you might specify in JCL on an IBM system. Switches are in general of the form "FILENAME/KEYWORD" or "/KEYWORD:VALUE" or just "/KEYWORD".

Among the more commonly used switches are:

/HELP prints a list of available switches.

/EDIT performs the same functions as an EDIT command inside of SPSS, allowing you to test your program without modifying it.

/SPACE:nnK where nn is the number of 1000-word blocks of space you need. It should be greater than 16 and not more than 50.

<file name>/SCRATCH
specifies the scratch file. This file can be very large. When only one statistical procedure and no SAVE FILE will be involved, you may avoid creating this file by saying: NUL:/SCRATCH

See the SPSS-10 guide (available as DOC:SPSS.DOC) for further details. Example:

```
*mytest.lst=mytest.sps,/edit/space:25k
```

If you run out of space in your directory and get a message about "Disk quota exceeded," you may want to use a system scratch directory for temporary workspace and storage.

30.9.5. Writing Files in SPSS

SPSS can output two types of data files: system files (binary) and raw data files. In the example above, line 1900 instructs SPSS to create a binary system file with the name MYFILE.FIL (.FIL is the standard file type given to SPSS system files). Note that this is the external name of the file i.e. the name by which the operating system knows it, rather than an internal SPSS "file name". On a later run, instead of reading in all the raw data and labels over again, you could simply include a command of the form:

```
GET FILE        MYFILE.FIL
```

You would want to have SPSS generate and save a system file only if you intend to make several runs using the same data and variable information. SPSS at LOTS will automatically create MYFILE.FIL in your disk area during a successful run of MYTEST.SPS above. Consult the SPSS manual for more information on SPSS system files.

Note that an SPSS system file will usually be larger than the .SPS and .DAT files combined.

Raw data files, the second type of file, also need to be named in the .SPS file. The following example uses MYTEST.SPS to create NEWFIL.DAT, a raw data set containing a subset of the original data file on disk.

```
RUN NAME        USING A GET FILE COMMAND
RAW OUTPUT UNIT NEWFIL.DAT
GET FILE        MYTEST.SPS
WRITE CASES     (3F1.0)V1 TO V3
FINISH
```

The RAW OUTPUT UNIT card must be used if you want an SPSS procedure to output calculations to a disk file, such as

a correlation or covariance matrix to be used as an input data file to a regression or factor run analysis run.

30.10. Using TSP

TSP (Time Series Processor) is an econometric package designed primarily for analyzing time series data. The following procedures are included in the package:

- Nonlinear and Linear Least Squares
- Multivariate Least Squares
- Nonlinear 2 and 3 Stage Least Squares
- Polynomial Distributed Lags
- Regressions with First Order Serially Correlated Errors

In addition, TSP has various matrix, graphics, conditional, and interpolation operators.

30.10.1. Documentation

The full TSP manual for Version 3.5 (IBM version) may be purchased from I.T.S. Document Sales.

30.10.2. Creating Your Program

To use TSP you must first create a file of the TSP statements. The input file name should contain at most six characters and the extension should have at most three characters. The extension "TSP" is suggested. PROB1.TSP, WHEAT.TSP, and A1.TSP are valid file specifications, while INCENDIARY.TSP and PROB.TSP1 are invalid. TSP statements must have at most 80 characters. Any statement which begins with a semicolon is interpreted as a comment.

The following is an example of a TSP program:

```
$$NAME MOORE$
LOAD$
GENR GNM1 = GNP(-1) - IMPT(-1)$
SMPL 1 9$
PRINT IMPT GNP RELP$
OLSQ IMPT C GNP RELP$
SMPL 2 9$
OLSQ IMPT C GNP RELP$ OLSQ IMPT C GNM1$
STOP$
END$

SMPL 1 9$
LOAD ID$
1948 1949 1950 1951 1952 1953 1954 1955 1956$
LOAD IMPT$
100 106 107 120 110 116 123 133 137$
LOAD GNP$ 100 104 106 111 111 115 120 124 126$
LOAD RELP$
100 99 110 126 113 103 102 103 98$
END$
```

30.10.3. Running a TSP Program

To run TSP give the command @TSP. You will be prompted for the name of the input file which you should have created previously. Next you will be asked if you want to see your output at your terminal; you should reply YES or NO. If you say NO, you will be prompted for an output file specification. As in the case of the input file, the output file name should have a maximum of six characters and its extension should have at most three characters. It is recommended that you use the same name as you used for the input file and the extension ".OUT".

If you wish to print the output file on the lineprinter, you will probably want to use the "/FILE:FORTRAN" switch so that the carriage controls (when to skip to a new page, and so on) are appropriately obeyed. For example, if the file is PROB1.OUT:

```
@print prob1.out/file:fortran
```

will have the desired effect.

31. Using Magnetic Tapes

This discussion consists of four sections. The first is a general discussion of tapes and tape formats, and provides general orientation on using tapes on the DEC-20. The second section documents some programs that are available for moving data via tapes to and from other computers. The third deals with the DUMPER tape handling program, which is useful for moving files to and from other DEC-20 computers, for saving your own files for later use on a DEC-20, and for restoring files saved on system backup tapes. The fourth section discusses in greater detail the formats you should use for moving files to other computers.

31.1. Why Use Tapes?

Tapes are a convenient and inexpensive means for storing a large amount of data. A single 2400' magnetic tape can store more than 100 million characters (some 32,000 pages), depending on the format chosen. Though data on tape is not as immediately available as data stored on disk, it has the advantage of being portable; you can take your tape with you to another computer center, or bring it back to your computer months after you have recorded it.

In particular, when you need to move data between computers that are not connected by the Stanford Ethernet, the easiest way is often on magnetic tape. Tapes are also useful if you have files that are too large to keep on your directory, or if you have files that you are not actively using.

If you want to move files between two computers on the Stanford Ethernet, then you should use the FTP program. FTP is quicker to use and more foolproof than carrying tapes between computers. As of this writing the only major Stanford computers that are not on the Ethernet are the IBM machines at I.T.S. and SLAC. See Chapter 26 for details on transferring files over the Ethernet.

If you need to use tapes but don't already have your own tape, you may purchase one from I.T.S. document sales at Forsythe Hall. Buy a 1200 foot or longer reel; for most purposes, you will do best to buy a standard 2400 foot reel (cost is about $18.00).

There are basically two tape handling programs that you might want to use to read or write data on a tape. The DUMPER program (see page 238) is useful for communicating with other DECSYSTEM-20 sites, or for making backup tapes that you plan to read your computer at some later date. Use the TAPEIO program (see page 235) for communicating with the Stanford I.T.S. 3081 systems, and with various other computers. In addition, there are a number of special purpose programs for reading and writing data in the various specialized tape formats recognized by other brands of computer.

31.2. Tape Fundamentals

A magnetic tape is physically very similar to the tape used in a home reel to reel tape recorder. However, the information storage technique is very different. On magnetic tapes, data is stored in "frames" i.e. 1-bit long slices of the tape. On 9-track tapes, each frame contains 9 bits, and so has room to store one 8-bit byte (normally one character) plus a parity bit for error detection. Tapes are described in terms of their "density", which is measured in "bits per inch" (bpi), and refers to the number of frames which can be packed into one inch of tape.

Tapes may be 7-track or 9-track, though 7-track is rapidly becoming very uncommon. Typical densities are 800 bpi, 1600 bpi, and 6250 bpi. All DEC-20's have 9-track tape drives capable of reading 1600 bpi tapes; check with the system staff if you are unsure whether your site can read 800 or 6250 bpi tapes. To read 7-track tapes and tapes written at other densities, you should take your tape to I.T.S. and copy it to a standard 9-track tape.

Data on a tape is organized into "physical records" or "blocks". Each block is simply a piece of tape with data recorded continuously on it. Blocks are separated from each other by 0.4 inch gaps, called "interrecord gaps". Tape drive hardware always works with blocks as single units. Thus it is possible to ask the tape drive to read a single block, and it will get exactly one block, no matter how many or few characters may be in the block. One file on tape, then, is made up of one

or more blocks.

Incidentally, many tape handling programs allow you to specify a block size, which is a guide to the operating system as to the maximum size blocks you want to write. Large blocks pack data much more efficiently than small ones: at 1600 bpi, to write 128,000 bytes (about 50 disk pages worth of data) onto a tape with a block size of 80 bytes requires 60 feet of tape; with a block size of 25600 the same data could be written onto 7 feet of tape, since less space would be wasted in interrecord gaps. On the other hand, the larger block size might not be readable by some small computers.

31.2.1. Normal DEC-20 Files: Stream-Oriented Files

On the DEC-20 files are "stream-oriented"; that is, files are simply an unstructured stream of bytes. In text files, ends of lines are marked by certain characters (usually CRLF, i.e. carriage-return followed by line feed), rather than by any special record structure. The operating system can copy files of this sort from one device to another, without regard for the physical property of the device. Block sizes and other physical structure of the device are handled transparently, since they are completely irrelevant to the logical structure of the file. To deal with files in this way, the COPY command in the EXEC is quite sufficient for all devices. It is perfectly legitimate to copy files to tape using the COPY command, though for some purposes you will want to use a special program.

To get a stream-oriented file in which characters are placed one character per frame, you must first tell the EXEC what bit packing technique to use, then copy the file to the magnetic tape device, MTAx:.[38]

```
@set tape format ansi
@copy myfile.dat mta0:
 MYFILE.DAT.3 => MTA0:MYFILE [OK]
```

31.2.2. Files on Unlabeled Tapes

When you use tape in the default manner (stream-oriented), the only structure you need to know about is files. That is, data can be thought of as a continuous stream of bytes or characters, with end of file marks separating individual files. If you copy three files to tape using the COPY command, you will get three sections of data:

```
beginning of                  logical end     physical end
|      tape                   of tape   |     of tape    |
V                                       V                V
-----------------------------------------\  \--------
| file 1  FOF  file 2  FOF  file 3  FOF EOF  /  /       |
| data    FOF  data    FOF  data    FOF FOF  \  \       |
-----------------------------------------/  /--------
```

EOF here represents an end of tape mark, which is a special code written on the tape and distinguishable from ordinary characters. Two successive EOFs represent the "logical end of tape" (LEOT).

When a tape is structured this way, there are no file names. You have to remember what is on the tape. Each disk file is stored as exactly one tape file, so you need to remember both the ordinal number of the file on the tape and its name.

You get to a particular file by using the EXEC's REWIND and SKIP commands. To read the third file on the tape mounted on MTA0: you might first make sure the tape is positioned at its beginning, then skip over the first two files:

```
@rewind mta0:
@skip mta0: 2
```

To add data at the end of a tape, you can use SKIP MTA0: LEOT to skip to the logical end of tape. This positions the tape between the two EOF marks. Thus if you write a new file what remains is one EOF mark to separate it from the file before it, and two new EOF marks after the new file on tape. It is almost impossible to read beyond the "logical end of tape". Thus, if you write a new file replacing the second file in the above example, the third file is lost.

---

[38] Various DEC-20's may have more than one tape drive. The first drive is called MTA0:, the second MTA1:, and so forth. The drives have their unit number labeled on the front.

31.2.3. DUMPER Tapes

The format described above is designed to be simple enough that any program can read and write from the tape. Because of this simplicity the tape does not have as much information about each file as there would be if the file were on disk. The file on disk has associated with it such information as its name, its last read date, possibly Edit line numbers, and so on.[39] For these reasons there is a special program, DUMPER, which writes tapes in such a way that all of the information from a disk file is saved on the tape. DUMPER also packs multiple disk files into a single tape file, and permits you to organize groups of disk files into "save sets."

This format allows files to be put back on disk exactly as they were originally. The price you pay for this greater amount of information is that the tape format is more complex. A normal applications program will not be able to read the tape. In practice, once a file is on a DUMPER tape, the only thing you can do with it is ask DUMPER to copy it back to disk. For most users this is a good tradeoff -- the vast majority of tapes on a DEC-20 are in DUMPER format. For further information on using DUMPER, see section 31.6, page 238.

31.2.4. Variations on Stream-Oriented Files

Two typical variations on the basic idea of stream files are common. First, a different bit packing technique can be used; second, ends of lines may be marked with different special characters.

As described above, stream-oriented files are designed for holding text. Each frame on the tape contains exactly one character. For storing binary data (e.g. EXE files, or SPSS system files), this is not appropriate, since one 36-bit computer word holds five 7-bit characters with 1 bit left over. Therefore, various other packing techniques are used. When using the `COPY` command with unlabeled tapes, you can choose what bit packing technique to use by means of the `SET TAPE FORMAT` command; for reading or writing files written on other systems you will almost always want format `ANSI-ASCII`; DUMPER, however, uses `CORE-DUMP` format so that it can record every bit of data stored in each word in a file.

Various computers which use stream format files use different characters to delimit ends of lines. For example, a tape written on a Data General Eclipse running the RDOS operating system uses `CR` (`CTRL/M`) instead of `CRLF`; Unix systems and DG Eclipses running AOS use `LF` (`CTRL/J`); Tenex systems use `CTRL/_`. If you want to read a tape written on one of these systems, the best technique is usually to copy it to disk using the `COPY` command, then to use a text editor[40] or write a special purpose program to translate the end of line characters into `CRLF`. When writing a stream-oriented tape to take to one of these systems, you will have to do the same sort of translation when you arrive there.

31.3. Record-Oriented File Structures

For historical reasons, most of the computer industry does not use stream oriented files. Instead, they use files where physical or logical record boundaries are used to define where lines begin and end. We will refer to these as "record oriented" file structures. When record boundaries are being used, carriage returns and line feeds are normally not present in the file.

When discussing record oriented files, the term "logical record" (or simply "record") is normally used to describe a line of text. The key question for each record oriented file type is how the blocks are broken up into logical records. Note that since record-oriented tapes are organized into lines, it makes little sense to use them on a DEC-20 for anything except text. They should not be used to store a binary file such as an SPSS system file or a `REL` (object) file.

---

[39] If the tape is labeled (see below), the file on tape has a file name and creation date associated with it. But a disk file has other data associated with it, such as the last read date and the number of times the file has been accessed.

[40] EDIT automatically translates LF to CRLF when it reads a file, but cannot cope with any other end of line sequence. The EMACS editor can be used to perform any of the changes described here.

31.3.1. The Record Formats

Recall that a tape is divided first into files, and then within files into physical blocks. The simplest record format puts one logical record in each block. However it turns out that this wastes tape. On 1600 bpi tape, an 80 character line takes about .05 inches. Since the inter-record gap is .4 inches, you can see that most of the tape would be wasted in inter-record gaps. Thus one normally puts more than one logical record into a block. This is referred to as "blocking". The various record formats basically result from using different conventions to separate the logical records that are combined into one block. Here are the major record formats. Formats are usually referred to by one- or two-letter abbreviations (the "B" in the abbreviations indicates that more than one record should be expected in each block). These abbreviations are shown below.

- Fixed format (FB) -- In this format every logical record (i.e. line) is made the same length. Lines that are too short are extended with blanks, and lines that are too long are truncated. The person writing the tape should choose a record length that will be long enough to contain the longest line in his file. The record length is often called the "logical record length" or LRECL. Usually several logical records are written in each block. The number of records per block is constant, which of course makes the size of the blocks constant also. The only exception is the last block in the file, which may be short if the number of lines in the file doesn't come out exactly. In order to decode a tape written this way, all the system needs to know is the LRECL. Since all records are the same length, it is very easy to break up a block into its logical records. Format FB is the simplest of the record-oriented formats, and is supported by virtually every computer in the world.

- Variable format (D and DB for ASCII files, V and VB for EBCDIC files) -- Variable format files are used when it is not desirable to extend all lines to be the same length. In this format, a special code is put at the beginning of each logical record to show how long that record is. This code is not part of the data. The operating system normally handles it automatically, and your program never sees it. Usually more than one logical record will be put into a block (although this is not necessary). Typically the user specifies a suggested block size. The operating system will keep packing logical records into a block until the block reaches the size requested. At that point it will start a new block. When records are blocked in this way, the system is able to separate the records by using the record length codes at the beginning of each. This format is not supported by as many computers. You should only use it if you are sure that the computer for which you are writing the tape can understand it.

- Spanned format (S) -- Spanned format is a slight variation on variable format. It is intended for use with files having a few very long lines. With variable format each record must fit into a single block. If you have a few very long records, this can lead to block sizes that would be impractically large. Spanned format uses slightly more complex length codes, which allow records to be broken across block boundaries. Thus a single record can be as long as you want. Spanned format tends to be the least well supported of the record types, and should generally be avoided.

- Undefined format (U) -- In undefined format, there is no structure specified for the blocks. Each tape read or write gets one block, and it is the program's business to create any length codes or other structure. One common use of format U is to put each logical record in one block. However various systems use it in various ways. You should not use format U unless you know what you are doing. In principle the DEC-20 stream oriented files can be considered format U. Since it is up to the user to supply whatever structure he wants, DEC simply defines that records are delimited by carriage returns followed by line feeds.

31.3.2. Other Formatting Information

In addition to the considerations just presented, one equally important is character coding. The DEC-20 uses the ASCII code. This code specifies the internal numerical code to be used to represent each letter and symbol. For example, when ASCII is being used the letter "A" is represented as the number 65. This code is an ANSI standard, and is used by most of the industry. However IBM and a few other manufacturers use EBCDIC. EBCDIC serves the same purpose as ASCII, but simply uses different numbers to encode the same letters; in it, "A" has the value 193. It is usually safe to write tapes in ASCII, since most computers can read and write ASCII, even if they use some other code internally. However it is

sometimes convenient to use EBCDIC when communicating with IBM equipment.

In addition to character code, the other important bit of data for text files is page formatting information, for example page marks. On the DEC-20, text pages are separated by special characters (CTRL/Ls) included in the file. Underlining is generally accomplished by including a carriage return (CTRL/M) with no line feed, followed by a second "line" containing just the underlining. Thus, the single line (for example from a Runoff output file):

```
I can underline words!
```

is typically represented as:

```
column 1
V
I can underline words!↑M
```

Other computer systems use different conventions for representing this sort of information. A common alternative representation uses Fortran-style carriage control characters; when writing a tape, this is termed "A" or "print" format: for each line, an extra character is included at the beginning describing whether the line should be placed on a new page (a "1"), on the next line of output (a blank), or overprinting the previous line (a "+"). Thus, on a tape written using Fortran carriage control characters, the example above would appear as two lines:

```
column 1
V
 I can underline words!
+
```

A tape file written in VB format with Fortran carriage control characters is said to be in "VBA" format.

31.3.3. Labeled Tapes

In order to read a record-oriented tape, the system must know which of the formats it is recorded in. In some cases it must also know other information. For example it must know the logical record length in order to decode fixed format tapes. It needs to know how formatting information like page marks is represented, and whether the tape is recorded in ASCII or EBCDIC. For this reason record-oriented tapes are normally labeled.[41] An unlabeled tape usually has a gummed label on the outside of the reel that identifies the contents and owner of the tape. A labeled tape, however, contains standardized information on the tape itself that describes the owner of the tape, and each data file on the tape.

Each file on the tape has a label. This label is invisible to you, but tells the operating system the name of the file, its format, the logical record length, the block size, and similar information. This allows systems to decode files automatically. In addition, at the beginning of the tape is a "volume label" which identifies the tape itself and allows the system to verify that you are using the right tape.

The system sees:

```
-------------------------------------------------------\  \------
| volume | head- | file 1 | trail- | head- | file 2 | trail-/  /     |
| label  | er 1  | data   | er 1   | er 2  | data   | er 2  \  \     |
-------------------------------------------------------/  /------
```

If you were to treat a labeled tape as unlabeled, then your data files would appear as the second, fifth, and so on files on the tape. In general, the nth data file would be tape file `3*n-1`.

[41] It is fairly common to have fixed format files without labels. In this case the person who writes the tape must tell the person who will read it that it is fixed format, and what the logical record length and block size are. However variable and spanned files are sufficiently sophisticated formats that they are usually used only with labeled tapes.

31.4. Loading a Tape on the Tape Drive

The tape drives on most of the DEC-20's at Stanford are self service, i.e., you must load and unload your tapes for yourself. This section describes how.

Before doing anything with a tape, you should use the EXEC command `ASSIGN MTAx:`, where x is the number of the tape drive you wish to use, e.g., "`MTA0:`" is a name of a tape drive. The `ASSIGN` command reserves a physical device for exclusive use by your job. Once you have the tape drive assigned, no job but yours can touch the tape. When you are done with the tape drive, unload the tape and give the command `DEASSIGN MTAx:` (where x is the unit number) to allow others to use the drive.

Mounting the tape on the tape drives is about as complicated as threading a home reel-to-reel tape recorder. But, rather than rely on the following directions, you might try to find someone who can show you. We are notorious for failing to provide comprehensible descriptions of the indescribable. If you are planning to write on the tape, a plastic write ring must be placed on the back side of the reel. If you are planning to read the tape only, remove any write ring that may be present.

31.4.1. Mounting a Tape on the TU78

A tape may be loaded onto the `TU78` either automatically or manually. To be loaded automatically, the tape must be a full 10.5" reel. If you are unsure about how full a tape reel is, use the manual procedures; it is very possible that you could damage your tape by using automatic loading when not appropriate.

1. Place your tape reel in position on the upper tape hub, rotate the reel until it slips easily into place, and press the reel-retaining actuator.

2. Close the drive front door.

3. Place the `AUTO/MAN` switch in the `AUTO` position.

4. Check that power is applied to the tape drive (power light on). If it is not, contact a consultant.

5. Press `RESET` switch.

6. Press `LOAD/REW` switch.

7. Press `ONLINE` switch.

The take-up reel will start to turn clockwise. The supply reel will turn counterclockwise a few turns and then rotate forward and eject the tape into the tape path. The tape will thread and wrap onto the take-up reel.

To thread a tape manually, follow the following procedure:

1. Place the supply reel in position on the upper hub, rotate it until it slips easily into place and press the reel-retaining actuator.

2. Place the `AUTO/MAN` switch in the `MAN` position.

3. Open the buffer door.

4. Manually place the tape leader between thread block number 1 and air bearing number 1 (see figure on the tape drive). Ensure that there is no tape slack or sag between the supply reel and thread block number 1, or damage to the tape may result.

5. Carefully close the buffer door. Make sure it is closed securely.

6. Close the transport front door.

7. Check that power is applied to the tape drive (power light is on).

8. Press the RESET switch.

9. Press the LOAD/REW switch.

10. Press the ONLINE switch.

If, at any time, you have doubts as to what you are doing with the tape drive, you should request assistance. This will save you time and trouble.

31.4.2. Mounting a Tape on the TU45

The supply reel goes on the top hub. Make sure the reel is facing in the proper direction. When you pull the tape, the supply reel should rotate clockwise. Lock the reel to the top hub by flipping the lever arrangement in the center of the hub.

Due to some sinister conspiracy, the tape drive was designed for 5-foot tall left-handed people. Grasp the end of the tape in your left hand. Pulling on the tape, move your hand over the tape path that is indicated by the arrows. Pass as close to each pulley and tape guide as possible. If you're skillful, you won't need your right hand until you get to winding the tape onto the takeup reel. Pull the end of the tape upwards, around the top of the takeup reel and down the right side. Holding the tape and the reel together, take several turns in the clockwise direction. Be careful not to touch the tape except in the few feet near the end. Close the transparent door, and press the LOAD-RESET button. The tape will begin to move slowly onto the takeup reel, then the tape drive will slurp the tape into the vacuum columns and advance forward to the load point. When all motion ceases, check the tape drive to see if the ON-LINE button is lit. If not, press it.

When you have finished loading the tape onto the drive, you are ready to run your program. Go back to your terminal and do so. When you are done, you may unload your tape, remove it from the tape drive, DEASSIGN the drive, and go about your business.

The easiest way to unload the tape is the EXEC command UNLOAD MTA0:. The tape should rewind to load point and then sedately unload itself.

31.5. Using TAPEIO and IBM-Compatible Tapes

The Stanford DEC-20's have two utilities to deal with record-oriented tapes. TAPEIO can be used to read and write tapes with a number of formats and label types. If you wish to label a tape, the TAPELABEL program will write an IBM standard label. A predecessor to TAPEIO named TOSCIP also exists, but use of TAPEIO is preferred. Currently the only source of hard copy documentation for these programs is this manual. The programs are internally documented so you can type "HELP" and "?" while running them.

31.5.1. General Information About TAPEIO

TAPEIO may be used to read and write tapes written in IBM format. The tape must be 9-track and contain only character data; this means that some of the things you *cannot* read or write are: SPSS save files, files written with Fortran unformatted I/O, compiled (REL files) or executable (EXE files) programs.

The principal advantage of TAPEIO is that it can read and write labeled tapes. Labeled tapes are particularly useful when you are transferring data to a computer whose tape reading preferences are unknown. The label provides a standard way to describe the format of the tape on the tape itself. It avoids the problem of errors arising when you try to read a tape and have forgotten what format it was written in.

The following commands are recognized: DEFAULT, READ, WRITE, QUIT, and HELP.

The READ command reads tape files to disk, the WRITE command writes disk files to tape, the QUIT command stops the program and returns you to the EXEC, the HELP command prints information about the program, and the DEFAULT command sets default switch values for later READ and WRITE commands.

The HELP command may be followed by an optional argument (a topic). For instance, HELP READ gives information relevant to the READ command, while HELP LRECL gives information pertaining to the use of the LRECL descriptor.

The WRITE command must be followed by the disk file to be copied to tape, and then the position on the tape where the file should be written, followed by the optional switches. The READ command must be followed by the position on the tape, then the name of the disk file to be created. The allowable tape positions are:

`END` Writes a file after the last file currently on the tape. Illegal on READ, since it does not make sense to read beyond the end of tape.

`LAST` Writes a file over the last file on the tape. Reads the last file on the tape.

`NEXT` Reads or writes a file at the next position on the tape. If the last file read was number 4, then a WRITE would replace file number 5.

n (for *n* an integer, e.g. 3) Reads or writes the *n*th file on the tape.

Many switches may be specified on DEFAULT, READ, or WRITE to control the data transfer. However, when reading labeled tapes most of these switches are unnecessary, since the information is contained in the label. When writing, the FORMAT switch is generally all you need.

The FORMAT switch allows you to specify certain useful combinations of LRECL, BLKSIZE, and RECFM in a single switch. The following are the recognized combinations:

Keyword	LRECL	BLKSIZ	RECFM	Notes
VCARD	250	30720	VB	cheapest for non-printing data
VPRINT	250	30720	VBA	cheapest for data to be printed
FCARD	80	30720	FB	best fixed length for data
FPRINT	80	30720	FBA	best fixed length for printing (if each line is LEQ 80 chars)
TCARD	80	8000	FB	TOSCIP's CARD format
TPRINT	133	7980	FB	TOSCIP's PRINT format
ANSI	2044	2048	DB	DEC's default labeled format (ASCII, keep CTRL/L)

The complete list of switches which may be specified in DEFAULT, READ and WRITE, and as topics in the HELP command is:

`/BLANKS` Specifies the treatment of blanks and nulls.

`/BLKSIZE` Size of physical tape records (ignored when reading labeled tapes).

`/BLOCK-PREFIX-LENGTH`
Length of an optional block prefix for ASCII tapes (READ only).

`/CASE` Specifies sensitivity to case (/CASE:UPPER means convert data to upper case).

`/COUNT` Number of logical records to be processed in this file.

`/CTRL-L` Specifies whether `CTRL-L`'s (page marks) are to be ignored.

/DATA-TYPE Specifies whether the tape is ASCII or EBCDIC (unlabeled tapes only).

/DENSITY Any of 800, 1600, or 6250 bpi (unlabeled tapes only).

/EOR Specifies the method of determining the end of the record.

/FORMAT Specifies LRECL, BLKSIZE and RECFM in a single descriptor.

/LEADING-TAPE-MARK
Specifies the presence of a leading tape mark (IBM DOS format).

/LINE-NUMS Specifies the handling of line numbers.

/LRECL Size of logical records on the tape.

/NULLS Specifies that nulls are to be flushed.

/PARITY The parity of the tape (don't ever specify this).

/RECFM The record format of the tape (ignored when reading labeled tapes).

/SKIP The number of logical records to skip over.

/TAB Specifies whether tabs should be converted to multiple spaces.

Almost all of these switches require an argument. For instance, "/skip:5" means that five records should be skipped and "/nulls:no" means that nulls are not to be flushed. The defaults for each of these can be obtained by "HELP VALUES". Note in particular, however, that if the tape is labeled TAPEIO will use by default the parameters specified in the label when it reads a file. For parameters not specified in the label, or for writing, switch values are "sticky". In other words, once the switch values are specified, they remain in effect during succeeding commands.

31.5.2. TAPEIO Examples

To start up TAPEIO and select MTA1::

```
@tapeio
TAPEIO version of 21-October-1982.
Type "HELP" for help.
TAPEIO>tape mta1:
[Assigned drive MTA1:]
TAPEIO>
```

To read the first file on an unlabeled tape (mounted on MTA0:) written with default parameters:

```
TAPEIO>read 1 bessel.for
```

To read the fifth and sixth files on a labeled tape:

```
TAPEIO>read 5,6  regres.sps,regres.dat
```

To write a file on to the end of a tape with VPRINT format, which is useful when writing a file containing formatted text to be reproduced on a printer:

```
TAPEIO>write chap3.mem end/format:vprint
```

To write three files on to the end of a tape in VCARD format, converting lower case to upper (useful for moving programs to other systems):[42]

```
TAPEIO>default /format:vcard/case:upper
TAPEIO>write ibeam.for,subrs.for,random.for end
```

[42] IBM Fortran insists that programs be in all upper case, hence the conversion. Note that Fortran programs to be moved to another system should not use the "initial tab" convention; statements should begin in column 7.

To write a file at the beginning of an unlabeled tape matching the IBM specification:

```
DCB=(DFN=2,RECFM=FB,LRECL=120,BLKSIZE=2400,OPTCD=Q)
```

you might specify the following:

```
TAPEIO>default/den:800/data:ascii
TAPEIO>default/recfm:fb/lrecl:120/blksize:2400
TAPEIO>write for21.dat 1
```

Tapes written with TAPEIO should almost always be labeled. If you have a new blank tape that you wish to write using TAPEIO, you should run the TAPELABEL program first.

31.5.3. TAPELABEL

You can initialize a blank tape or replace an existing tape label on a magnetic tape with the program TAPELABEL.

To use TAPELABEL, mount your tape, assign the tape drive using the EXEC ASSIGN command and give the EXEC command TAPELABEL. The program will ask you a few questions about the tape and the label; when in doubt type a question mark.

For files being moved to I.T.S. we recommend that, rather than using TAPELABEL, you have I.T.S. initialize your tape, then *temporarily* check it out. This will insure that the volume number recorded on the internal label matches the reel number on the physical label pasted to the outside of the tape. If you do decide to use TAPELABEL, we recommend for most purposes specifying an IBM standard (SL) label, a density of 1600 bytes per inch, and a volume number that does *not* start with the characters "AU", "AS", "AX", "ES", "EU", or "EX".

```
@tapelabel
TAPELABEL (Version 1.03)

   This program labels a tape.  The previous contents of the tape are
destroyed!  Type '?' for help if you don't understand any of the questions
you are asked.

Tape UNIT number?
Volume serial number?
Owner name?
Sl or Al label?
Tape density?
 [Done]
@
```

If you answer any of these questions by typing a RETURN, you will get the default value, which is generally exactly what you want. The defaults are:

Unit number	0	Which tape drive you are using.
Volume	000001	Any 6 alphanumeric character string.
Owner	your user name	(e.g. F.FRANK) Any 10 character string.
Label	SL	I.E. IBM EBCDIC label; use AL for ANSI labels.
Density	1600 bpi	May be 800 or 1600.

31.6. Using DUMPER to Save or Restore Files

DUMPER is a DEC utility useful for saving and restoring disk files on magnetic tape. You can use it to recover files saved on system backup tapes, or to conveniently extend your disk storage with your own tape. DUMPER is very convenient to use for archiving since it knows about files on tape by name, but it uses a special tape format known only to DEC-10 and DEC-20 computers, so don't use it if you want to move files to another computer; use TAPEIO instead.

31.6.1. Documentation

For a more complete description of the DUMPER commands, see the DEC manual *DECSYSTEM-20 User's Guide*, Part II, or DOC:DUMPER.TUT on line.

31.6.2. Running DUMPER

DUMPER has a rather large set of commands. However, only a small subset of the DUMPER commands are needed in most cases. The frequently needed commands are described here.

To use DUMPER, you should first obtain exclusive use of the tape drive by assigning it, then mount your tape and run DUMPER. In the following example, we use DUMPER to display the directory information for every file on the tape mounted on drive MTA0: When you are done, EXIT from DUMPER, unload your tape, and deassign the tape drive.

```
Find your tape, and log in.

@assign mta0:

Mount your tape now.

@dumper
DUMPER>tape mta0:
DUMPER>print tty:
   . . .

DUMPER>unload
DUMPER>exit

Remove your tape from the tape drive before deassigning

@deassign mta0:
@
```

The PRINT command in DUMPER may be used to output a directory of the tape to an arbitrary file by replacing the "tty:" in this example by some other file specification.

If you are planning to use DUMPER to save your own files, you will need to have a tape of your own. You will need a tape that is certified for 9-track operation at a minimum of 1600 bpi. Tapes 1200 feet long and longer are preferred.

31.6.3. Restoring Files from Tape to Disk

The RESTORE command in DUMPER is analogous to an EXEC COPY command. It creates a copy of a file from tape onto disk. The simplest form of the RESTORE command is:

```
DUMPER>restore filename.ext
```

or

```
DUMPER>restore file1.abc,file2.ghj,file3.lmn
```

This command restores the specified files from the tape. You must not have files with the same name or DUMPER will not restore them from the tape. They must be stored under your directory name on the tape and they will be restored to your directory on disk. To restore all files saved under your directory, say:

```
DUMPER>restore *.*.*
```

If you wish, you may specify a new name for the file(s) you are restoring. In the following example, the user changes the file type of a file as he restores it:

```
DUMPER>restore prob1.pas prob1.pgo
```

31.6.4. Tape Positioning Commands

DUMPER tapes are divided into *save sets*, each containing one or more (usually many) named disk files. Each time a SAVE command is used to transfer files from disk onto a tape, a new save set is made. For system archives, an entire dump (perhaps a dozen tapes) is usually only a single save set. For your own tapes, you will probably want to have several save sets on the tape (perhaps one for each time you save files).

Note that you can only write at the logical end of the tape. Thus, if you have three save sets on the tape, and you skip the first then save a new set of files where the second used to be, the third save set is irretrievably lost.

```
DUMPER>skip 2
```

This command skips the specified number of "save sets". Use the skip command in conjunction with a directory listing of the tape to specify the save set from which to restore files.

```
DUMPER>eot
```

This skips to the end of the tape, after the last existing save set, so that you can add new files in a new save set.

```
DUMPER>rewind
```

This rewinds the tape so that you are once again at the beginning.

You should note that all of the disk files and all of the save sets on a tape are normally packed into a single tape file. Thus, the SKIP command in DUMPER is *not* the same as the EXEC command `SKIP MTA0:`.

31.6.5. Saving Disk Files on Tape

```
DUMPER>save file1.nam,file5.name,file7.ext
```

This saves the specified files onto the tape. Be sure you use an EOT command or do an appropriate number of SKIPs before doing this or you may write over files you meant to keep. Also, this will work only if you have a write ring in the tape while it's on the drive. To save all your files, you might say:

```
DUMPER>eot
DUMPER>save *.*
```

31.6.6. Additional Notes

Some of the above commands have various extra features; in particular you may sometimes want to specify an explicit directory rather than using the current connected directory. DUMPER is a little funny about the way it defaults directories, so if you are using any directory anywhere that is not your current connected directory, or any structure other than `PS:`, you are advised to specify a directory or structure everywhere it is possible, and to specify a destination (what the file will be called when you get it on the disk) for each file group restored, so that you can specify the directory.

For if you are logged in as `F.FRANK`, don't use:

```
DUMPER>restore <g.gazorn>dates.txt
```

but rather:

```
DUMPER>restore <g.gazorn>dates.txt  <f.frank>dates.txt
```

because once you specify `<G.GAZORN>`, *it* becomes the default place DUMPER tries to restore the file to, instead of your logged in directory.

If you are using DUMPER to move a file to or from a DECsystem-10 computer, you should give the INTERCHANGE command to DUMPER before saving or restoring files.

31.7. Transferring Files to and from Other Sites

The DEC-20's at Stanford can write tapes readable by most other sites, and can read most tapes written for export from other sites. The tape must be nine-track and, depending on the type of tape drive, can be any of 800, 1600 or 6250 bpi.

If you are writing a tape on another DEC-20 for use at Stanford or if you are writing a tape at Stanford that you are sure is only going to be used with a DEC-20 (or DEC-10) computer, you should use the DUMPER program (see page 238). For most other computers, reading and writing IBM style tapes with TAPEIO is probably best.

31.7.1. Importing Foreign Tapes

When reading a tape from another computer, the more information you have on the format the better. If the tape has a standard internal label, then the system will figure out the format for you. All you need to do is give a READ command to TAPEIO (see section 31.5) specifying which file on the tape you want, and what you want to call the file on disk. If the tape is not labeled, then you must tell TAPEIO what the format of the tape is. If you don't know the format, you might try the TAPELOOK program (see page 241) first, or ask a consultant for assistance.

In general, when writing a tape at a non-DEC-20 computer to bring to a DEC-20, you should try to have it written in a standard IBM format if possible. Obtain as much information as possible about your tape, for example:

On what type of computer was the tape written? What operating system?
At what density was the tape written?
Does the tape have an internal label? What kind?
Does the tape contain EBCDIC or ASCII characters?
What was the record format (e.g. F, FB, D, VB)?
What was the block size (physical record length)?
What is the record length (logical record length or line length)?
How are ends of lines indicated on the tape?
How many files are there on the tape?

If you are writing a tape at an IBM OS system, you should write a standard labeled tape at 1600 bpi, in either FB or VB format. On other systems (except as noted in section 31.7.7), the simpler the format the better. Best is probably Fixed Block format, where each line of data is padded with spaces to the same length (often 80 characters), and where each block contains the same number of lines. You should avoid tape formats that are designed for system backup, since such formats usually contain data relevant only to the file system of the particular computer on which the data was originally stored.

You should be careful to avoid sending or receiving a tape written in a special machine-dependent format or code. For example, Wylbur Edit format at I.T.S. is machine dependent and should never be used. IBM libraries (partitioned data sets) are also non-portable, though you may write individual members of a library onto your tape as separate files. Similarly, the internal formats of SPSS and BMDP system files vary greatly from machine to machine, and hence are not generally portable.

Whenever you are receiving or sending a computer tape, it is useful to have a copy of the run of the program that created the tape (at I.T.S., the batch job log; on a DEC-20, a list of the commands given or a PHOTO of the terminal session), as well as a listing of some of the data on the tape. This will make verification that the data has been transferred successfully much easier.

31.7.2. Reading Mystery Tapes -- TAPELOOK

If you are attempting to read a tape from another computer facility but you don't know what format it was written in, you might be able to get some hints from TAPELOOK. TAPELOOK is a program which attempts to read a tape assuming a variety of formats and densities, to interpret any labels the tape may have, and then to display the first few lines of each file on the tape. TAPELOOK is useful when you are not sure what is on your tape or are not sure what parameters were

used to write the tape. TAPELOOK will determine whether or not the tape is labeled, and will print out the first part of each file on the tape.

Run TAPELOOK by typing TAPELOOK to the EXEC. To the prompt "OPTIONS?", you can respond with a carriage return to get the default behavior, or with any of the following:

STARTING (FILE) n	n = *decimal starting file number, default 1*
ENDING (FILE) n	n = *decimal ending file number, default 9999*
OUTPUT (FILE) file-name	file-name = *file name for output, default* TTY:
PRINT (# OF BYTES) n	n = *number of bytes of each file to print, default 80*
LENGTH (OF TAPE) n	n = *tape length in feet, default 2400*
TAPE (UNIT ID) device	device = *tape drive name, default* MTA0:

For example, the following command sequence sends the output to the file TAPE.DIR and prints the first 100 bytes of each file.

```
@tapelook
OPTIONS? output tape.dir print 100
```

The defaults are START 1, END 99999, OUTPUT TTY:, PRINT 80, LENGTH 2400, and TAPE MTA0:. Type only a RETURN if these are acceptable.

31.7.3. Choosing a Format for Exporting Tapes

If you want to be able to move your files to another computer, you should try to find out what formats that computer supports. If you know what parameters the destination site prefers, you can specify those when running TAPEIO. In the absence of information from the other computer, use the guidelines presented in this section or contact a staff member.

It is usually best to use labeled tapes, since it is more convenient to deal with tapes where the file names and formats are known automatically. However, some computers don't yet support labeled tapes, and treat labels just as additional files on the tape. Generally an unlabeled ASCII tape written in fixed format can be handled by any system.

If you have no other information, then it is a good idea to write a standard labeled tape using TAPEIO and the default parameters. This produces an ASCII tape with a record format of FB, a block size of 8000, a record length of 80, and a density of 1600 bytes per inch.

If you know that the computer you are concerned with can handle them, here are some guidelines on more sophisticated record-oriented formats:

- Always use labeled tapes.
- If the lines in the file are of similar length, use fixed format, with a record length long enough to accommodate your longest line, and enough records combined into one block to make fairly large blocks (e.g. 10000 characters for 1600 bpi tapes).
- If the lines are of different lengths use variable format, with fairly large blocks (again about 10000 characters for 1600 bpi tapes).
- If you are writing a tape to be taken to a minicomputer, use smaller block sizes, for example 2048.

As mentioned above, you should not try to send non-standard representations of data unless you are certain that the receiving computer can read them successfully in that form. Do not send binary files (e.g. SPSS system files, REL or EXE files) unless you are exporting a tape to another DEC-20 site using DUMPER.

31.7.4. Moving to and from the Information Technology Services (I.T.S)

The first thing you will need if you are going to transfer files between I.T.S.[43] and a DEC-20 is a tape. You may purchase a tape from I.T.S. document sales in the Forsythe building. Tapes of various lengths are available, but the 2400 foot length is the most convienient for use on a DEC-20.

You will also need a I.T.S. account; contact I.T.S. accounting for information.

31.7.5. Tape Transfer from I.T.S.

If you are going to be transferring data from I.T.S. to a DEC-20 you should have I.T.S. initialize the tape with a standard label by submitting the tape with the appropriate form at the Forsythe Hall information desk. Then log in under WYLBUR, and give the WYLBUR command `EXEC FROM LIB#TOLOTS PUBLIC` to write the files to tape. Then you can check out the tape and bring it to your DECSYSTEM-20. When you get here assign the tape drive with the EXEC `ASSIGN` command and run TAPEIO. TAPEIO will read the label and figure out the format of the data on the tape. For example if you want to read the first two files on your tape, you would say:

```
@tapeio
TAPEIO>read 1 file1.sps
TAPEIO>read 2 file2.dat
TAPEIO>quit
```

It is advisable not to scratch your file at I.T.S. until after you have verified that the file can be read and used successfully on the DEC-20.

31.7.6. Tape Transfer to I.T.S.

If you are transferring data from a DEC-20 to I.T.S., it is probably a good idea to have the tape initialized at I.T.S. with a standard label, and a density of 1600 bpi, then check the tape out temporarily and bring it your system. Make sure that you specify *temporary* on the check out form! Alternatively, if the tape does not already have a standard label you can initialize the tape on the DEC-20 by running the TAPELABEL program. However, if you use TAPELABEL to initialize your tape then I.T.S. will treat the tape as "foreign labeled," and you will need to specify the NOVOLCHECK option of IOPROGM when reading it at I.T.S..

On the DEC-20 assign the tape drive and run TAPEIO. Use the TAPE command to specify which tape drive you will be using. Then use the WRITE command. For instance, to write the files FUN.SPS and FUN.DAT on to the first and second files of the tape in VCARD format you would say:

```
TAPEIO>write fun.sps 1/format:vcard
TAPEIO>write fun.dat next/format:vcard
```

There are two reasonable formats for your data, depending on whether you plan to edit the data at I.T.S. or just print it. If you are in doubt about which format you want, specify VPRINT, since it is easy to remove unnecessary carriage control information from a file which has it, but almost impossible to replace information which has been lost.

- If you plan to print your file directly, then use /FORMAT:VPRINT (see page 233 and page 236).
- If you plan to edit the data using WYLBUR or plan to use it as input to a program, then you probably want /FORMAT:VCARD (see page 236).

```
TAPEIO>write thesis.mem 1/format vprint
```

After you have written the tape check it back in at the I.T.S. information desk and give the WYLBUR command `EXEC FROM LIB#FROMLOTS PUBLIC` to read the files onto a I.T.S. public disk volume.

---

[43] A collection of IBM machines run by an organization that now calls itself Stanford Information Technology Services. It has been previously called C.I.T. and S.C.I.P.

If you are planning on using the user-operated tape drive at Forsythe Hall, beware that it supports a *maximum* blocksize of 4100. Using TAPEIO with the VCARD format produces tapes that cannot be read in on this tape drive. Instead you must check in your tape at the Information Window for mounting by the operator. It does work, however, to set the blocksize to 4000 and use the VB record format with a logical record length of 250.

Do not delete your file(s) on the DEC-20 until you have made certain that the file you read at I.T.S. contans the data you want.

31.7.7. Formats Accepted by Various Other Computers

This section attempts to summarize information about the formats that various computers understand. When possible, it indicates what software on these computers can be used for each format. You will have to consult the documentation for the individual computers for details.

Typically each computer will support some set of record-oriented formats. This will be the best way to communicate to it from another computer. However, most computers also have some special complex format for saving disk files with as much information about them as possible; the DUMPER format described above is an example of such an internal format.

DEC-20

Unless you know otherwise, you should assume that DEC-20 systems have 1600 bpi tape drives. The preferred format for DEC-20 systems is DUMPER. Systems may be configured to include "Tape Labeling", in which case it is much more difficult to read and write unlabeled tapes.

- Stream-oriented data: by default uses a special encoding of characters onto the tape, so that such tapes are only useful with another DEC-20 or DEC-10 computer; this default may be over-ridden with the `SET TAPE FORMAT` command. Software: to read or write unlabeled tapes containing standard stream oriented data, first give the EXEC command `SET TAPE FORMAT ANSI`, then use the `COPY` command.

- Unlabeled tapes, record-oriented data: no standard system commands for this, but most programming languages can deal with fixed format easily. The programs described below for labeled tapes can be also be used for fixed formats.

- Labeled tapes: if "tape labeling" is enabled, the system allows you to read ANSI and EBCDIC labeled tapes, and write ANSI labeled tapes, by means of the `COPY` command or via programs. At sites where tape labeling is disabled in the operating system, you can use programs such as TAPEIO (see section 31.5). Other programs which may be available at other sites if TAPEIO is not include TOSCIP, CHANGE (not available at Stanford), MTU, and VAXTAP.

- Internal format: the internal format for saving disk files is DUMPER (see section 31.6).

DEC-10

Unless you know otherwise, you should assume that DEC-10 sites have 800 bpi tape drives. DEC writes distribution tapes for DEC-10's at 800 bpi, so all sites should be able to read it. More recent sites all have 1600 bpi capacity also. The prefered format for DEC-10 systems is BACKUP (which can be produced by DUMPER, as explained below).

- Stream-oriented data: uses a special encoding of characters onto the tape, so that such tapes are only useful with another DEC-20 or DEC-10 computer. Software: `COPY` command, all programming languages. At the DEC-20 end, do not specify `SET TAPE FORMAT ANSI`.

- Unlabeled tapes, record-oriented data: no standard system commands for this, but most programming languages can deal with fixed format easily. The program CHANGE can handle all formats of unlabeled EBCDIC tapes, and fixed format ASCII tapes.

- Labeled tapes: most sites do not yet have support for labeled tapes in their system, although DEC is working on support which will be the same as that on the DEC-20. The program CHANGE can be used to read and write all formats of EBCDIC labeled tapes. It can also read and write fixed format files on ANSI labeled tapes.

- Internal format: there are several internal formats, of varying age. The oldest is FAILSAFE. We have a copy of the program for reading this format on the DEC-20. The current format is BACKUP. On DEC-20's, this format is handled by DUMPER, if you use the INTERCHANGE command before the SAVE or RESTORE commands. To specify 800 bpi, you should then say DENSITY 800.

VAX/VMS You should assume that VAX sites all have 1600 bpi capability. The preferred format for VAX is record-oriented ANSI labeled tapes.

- Stream-oriented data should be possible, but the bit encoding on tape is different from the default on the DEC-20.

- Labeled tapes: ANSI labeled tapes are well supported in all formats. As far as we know, EBCDIC labeled tapes are not. You can write ANSI labeled tapes on a DEC-20 by means of TAPEIO or the VAXTAP program.

- Unlabeled tapes: fixed format tapes can be handled by properly written programs, but are not easy. You would have to do EBCDIC conversion yourself.

- Internal format: DSC. This is not useful for the DEC-20.

IBM/360, 370, and later upgrades, using OS

1600 bpi is probably the safest assumption, though many sites now default to 6250 bpi. The preferred format for transferring files to or from an IBM site is a standard IBM labeled tape in almost any format (see section 31.7.6). For tape <transfer to and from the Stanford I.T.S. 3081, see section 31.7.4, page 243.

- Stream data: difficult to impossible.

- All record-oriented formats are well supported from both higher-level languages and utilities.

- Labeled tapes: may use any of the formats. Both ANSI and EBCDIC labels are supported. However, on ANSI tapes, format DB and S are somewhat nonstandard, so these formats may not be useful for communications with the DEC-20.

- Internal format: a number. IEHMOVE, IEBCOPY, and IEBGENR are common ones. These are not useful for the DEC-20.

UNIX (various machines)

For PDP-11's you should assume 800 bpi, for VAXes 1600 bpi. The preferred format is currently TP.

- Stream: unfortunately the bit packing is different from that used by default on the DEC-20. You might try the DEC-20 program 11TAPE.

- Record-oriented formats, labeled or unlabeled: difficult if not impossible.

- Internal formats: most systems support TP format, though modern Unix systems use TAR instead for most tape handling. There is a program TP on the DEC-20 to deal with tapes in TP format. It is well supported and easy to use.

RSX-11M (PDP-11)

The preferred format is stream.

- Stream I/O: the RSX utility PIP can read and write stream-oriented data.
- Record-oriented formats: some large systems are configured to include support for ANSI labeled tapes. If the system is not so configured, handling such formats is very difficult.
- Internal formats: unknown.

Hewlett-Packard 3000

The preferred format is ANSI labeled tapes, with record and block sizes that are even.

- Stream I/O: probably difficult.
- Record-oriented formats: these are supported by the standard utility (FCOPY) and programmming languages. All formats are supported, but for labeled and unlabeled tapes the record and block sizes must be even. ANSI labeled tapes can be read and written, EBCDIC tapes read only.
- Internal format: SAVE, not useful with the DEC-20.

Data General NOVA and ECLIPSE

The preferred format is unlabeled ASCII stream.

- Stream I/O: this is the most common format available on DG computers. Use the EXEC commands `SET TAPE FORMAT ANSI` and the `COPY` command. Note, however, that end of line is indicated by `CR` under RDOS and by `LF` under AOS. Thus, on writing you will need to replace the `CRLF` that the DEC-20 writes with the appropriate line terminator; on reading a tape on a DEC-20, you must replace the `CR` or `LF` with a `CRLF` sequence.
- Record-oriented formats: ANSI and EBCDIC labeled tapes in all formats are claimed to be supported under AOS version 3.2 and later, but there is some doubt whether they actually work. In particular, use of labeled tapes requires the presence of an operator on the Eclipse system.
- Internal format: various different formats under different operating systems. The DEC-20's can write a variant of RDOS format (work in progress. Contact a staff member for details), which is readable under AOS via the X RDOS LOAD command.

CDC Cyber/6000 Series, using SCOPE/KRONOS/NOS

800 bpi is probably the safest assumption, although most sites will have 1600 bpi capability. The preferred format is ASCII encoded ANSI labeled tapes, with record and block sizes that are even. Block sizes that are relatively small (under four thousand) are recommended.

- Stream I/O: Difficult or impossible for the user to control at the CDC site. Also, the CDC character set (Display Code), and the rather non-standard use of CDC's End of Record (EOR), End of File (EOF) and End of Information (EOI) marks in disk files makes reading CDC stream data at non-CDC sites very difficult.
- Unlabeled tapes, record-oriented data: The command language allows for unlabeled tapes, but their use is not recommended.
- Record-oriented formats: ASCII and EBCDIC ANSI labeled tapes in most formats are claimed to be supported with the LABEL control card under SCOPE/KRONOS/NOS. A format which seems to work uses the ASCII character set, a logical record length of 80, a

fixed block format and a block size of 2400.

- Internal format: All internal formats should be avoided. The Cyber/6000 Series machines use a non-standard six bit character set known as Display Code and almost all of the internal formatting programs use this character set. The peculiarities of EORs, EOFs and EOIs described above also cause problems. To make things worse, many versions of Display Code exist, and there is no program at Stanford to translate from any of them. In particular, the CDC file support utilities MODIFY and UPDATE should not be used on your programs. These utilities will insert a large amount of editing history in your files, and will make your tape totally unreadable.

Appendix A
EDIT Character Set

	0	1	2	3	4	5	6	7	
	CTRL/@	CTRL/A	CTRL/B	CTRL/C	CTRL/D	CTRL/E	CTRL/F	CTRL/G	(a)
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	(b)
	none	'!	'"	'#	'$	'%	'&	'?	(c)
	CTRL/H	CTRL/I	CTRL/J	CTRL/K	CTRL/L	CTRL/M	CTRL/N	CTRL/O	(a)
01	BS	HT	LF	VT	FF	CR	SO	SI	(b)
	'(						')	'*	(c)
	CTRL/P	CTRL/Q	CTRL/R	CTRL/S	CTRL/T	CTRL/U	CTRL/V	CTRL/W	(a)
02	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	(b)
	'+	',	'-	'.	'/	'0	'1	'2	(c)
	CTRL/X	CTRL/Y	CTRL/Z	CTRL/[	CTRL/\	CTRL/]	CTRL/^	CTRL/_	(a)
03	CAN	EM	SUB	ESC	FS	GS	RS	US	(b)
	'9	'6	'4	'=	'<	'>	'7	'8	(c)
04	Space	!	"	#	$	%	&	'	(b)
								''	(c)
05	(	)	*	+	,	-	.	/	(b)
06	0	1	2	3	4	5	6	7	(b)
07	8	9	:	;	<	=	>	?	(b)
10	@	A	B	C	D	E	F	G	(b)
11	H	I	J	K	L	M	N	O	(b)
12	P	Q	R	S	T	U	V	W	(b)
13	X	Y	Z	[	\	]	^	_	(b)
14	`	a	b	c	d	e	f	g	(b)
	'@	'A	'B	'C	'D	'E	'F	'G	(c)
15	h	i	j	k	l	m	n	o	(b)
	'H	'I	'J	'K	'L	'M	'N	'O	(c)
16	p	q	r	s	t	u	v	w	(b)
	'P	'Q	'R	'S	'T	'U	'V	'W	(c)
17	x	y	z	{	|	~	}	rubout	(b)
	'X	'Y	'Z	'[	':	']	'3	'\	(c)

(a) Control Code Representation
(b) Character Name or Graphic
(c) EDIT C128 Mode Representation

Appendix B
System Program Summary

The following is an incomplete list of the system programs available on the Stanford DEC-20's as of September 1984, with very brief explanations of each. Note that the set of system programs changes rapidly as new programs are written and old ones are superseded. More detailed descriptions of most of these programs may be obtained by giving a HELP command. For a current list of what programs are available, give the EXEC command INFORMATION LOGICAL-NAME SYS:, then give DIRECTORY commands for the directories currently in the search list for SYS:.

11TAPE	Read and write tapes for a PDP-11 with a DEC operating system.
2COL	Reformat text into 2 columns for listing.
AID	A programmable calculator. (Say HELP AID.)
ALLOC	Shows your console allocation as of the present moment (LOTS only).
APLSF	The APL interpreter
ASM6809	Cross assembler for the Motorola 6809 processor.
ATSIGN	Source cross-reference listing maker. (Say HELP ATSIGN.)
BASIC	Basically simple programming language. (See page 87.)
BBOARD	Read the online bulletin board and gripes file.
BEMACS	A stripped down, beginner's version of the EMACS editor.
BMDP	Most of the BMD and BMDP-1981 series. (Say HELP BMDP.)
CALC	Another calculator.
CHAT	Makes two and three way terminal links less confusing.
CLEAN	Program to help you clean up your directory.
COOKIE	Print a fortune-cookie message.
CREF	Produce a cross-reference listing.
CRYPT	Encode files for security. (See HELP CRYPT.)
DBEDIT	LOTS only. Change or update information about you or your use of LOTS.
DFIND	Locate a string in a file.
DLINE	The display queueing program that runs on each queueing terminal.
DIRED	Directory editor. Works only on display terminals.
DOIT	Like COOKIE only different.
DOVER	Sends a file the Dover laser printer at Computer Science.
DPYPAS	Display package for Pascal, FORTRAN, and assembly programs.
DSR	DEC Standard Runoff. (Say HELP DSR.)
DUMPER	Program to move files to or from tape. (See page 238.)

DYN3	Dynamo III simulation language.
DYNAMO	Dynamo II simulation language.
EDIT	The standard system text editor.
EMACS	Powerful display editor originally from MIT (TECO based).
ETEACH	EMACS tutorial.
FAIL	Stanford DEC-10/20 assembler. Faster than MACRO.
FILCOM	Compare two files, and note all differences. (See also SRCCOM.)
FILDDT	File DDT.
FILUSE	Show how many times files have been accessed.
FIND	Locate a string in a file. Useful for address books, etc.
FINGER	Give information on users.
FORDDT	Fortran debugger.
FORSTA	Program for timing and tuning Fortran programs.
FORTRA	The Fortran compiler. (See page 143.)
FTP	File transfer program for the SU-Net and Arpanet.
GLOB	Generate listing of globals in binary libraries.
GRIPE	Send a suggestion to the system. (Say HELP GRIPE.)
GROUPE	Records and maintains information on user and directory groups.
H2PLOT	Easy to use plotting program.
HELP	Extended information on various system features.
HELPMAINT	Edit the keyword table used by the HELP program.
IDDT	Version of DDT that runs in a superior fork.
IPSTAT	Shows status of TCP network connections on the system.
JSYS	Allows on-line search of the JSYS or Monitor Calls manual.
KERMIT	Allows file transfers between a micro-computer and a DEC-20.
LINDO	Linear and integer programming problem solver.
LINK	Load REL files into memory.
LISP	Student version of MACLISP. (Say HELP LISP.)
LK	Shows which terminals are linked (TALK, PHOTO, etc.).
MACLISP	List-oriented programming language. (Say HELP LISP.)
MACRO	DEC's standard assembler; five times slower than FAIL.
MAKLIB	Turn a file into a format printable by a Canon laser printer.
MAKLIB	Create a program library.

MATLAB	Matrix manipulation program.
MERLIN	A program for moving renaming and copying entire directories.
MIDAS	MIT's assembly language.
MINILP	Simple linear program solver.
MINITAB	Interactive statistical package.
MM	The standard mail reader and sender. (See page 157.)
MMAILBOX	Manages the mail forwarding database.
NOVICE	Interactive introduction to the system.
OPR	Privileged program which permits OPERATOR to control system.
PA1050	The TOPS-10 to TOPS-20 compatibility package.
PASCAL	ALGOL-like language with user-defined data types.
PASSGO	Fast load-and-go version of the LOTS Pascal compiler.
PCREF	Format and cross reference Pascal programs.
PERUSE	Scan a large file, typing only selected pages.
PFORM	Reformat (prettyprint) a Pascal program.
PHOTO	Make a record of your terminal session. (See page 193).
PIP	TOPS-10 "Peripheral Interchange Program"; obsolete mostly.
PIVOT	Matrix pivoting.
PLOT	A plotting program. Suitable for use with the Dover and Imagen (Canon) laser printers.
PPL	Extensible, conversational programming language.
PTYCON	Pseudo-terminal control program.
PUPECHO	Used to check if a SU-Net host is responding to Ethernet traffic.
REMIND	Programmable appointment/event/trivia reminder. (Say HELP REMIND.)
RENBR	Renumber or reformat your FORTRAN program. (See page 143.)
REPLY	Send a reply to the last terminal message you received.
REV	Clean up your directory or change characteristics of files.
RUNOFF	Document preparation program. Use DSR instead.
SAIL	Our local Algol dialect.
SCSS	Conversational SPSS.
SDDT	Assembly language debugger (DDT) with MONSYM symbols.
SEND	Send a message to a logged in user.
SIMULA	Algol-like language especially suited for simulation.
SNOBOL	String-processing programming language.

SPEAR	System performance evaluation and reporting tool.
SPELL	Spelling correcter program. (Say HELP SPELL.)
SPSS	Statistical Package for the Social Sciences.
SRCCOM	Compare two source files.
SSORT	Fast string sorting program.
SUBDIR	Create and modify subdirectories.
SYSDPY	System status display.
TAGS	Generate an EMACS tags table.
TAPEIO	Read and write IBM labelled and unlabelled tapes.
TAPELABEL	Write a standard IBM or ASCII label on a tape.
TAPELOOK	Quick examination of the contents of a mystery tape.
TECO	Very powerful and concise text editor.
TELNET	Connect and log onto other hosts (computers) on a network.
TIPSTAT	Check the status of an Ethertip.
TN	Same as TELNET.
TSP	Time Series Processor. Statistical analysis package.
TTYINI	Set terminal characteristics on login.
TTYLOC	Customize your terminal location as displayed by FINGER.
TVEDIT	A display oriented text editor.
UDDT	DDT without MONSYM symbols. (See SDDT.)
VAXTAP	Read and write files on tape for VAX computers.
WATCH	Obtain data useful for system performance analysis.
WHAT	Type out last SEND or TO message in case it was obscured.
WIZARDS	Show all the privileged users currently logged in.
XSEARCH	Find a string in a file. (Say HELP XSEARCH.)
ZED	Another TECO based display editor. Developed at Stanford.

Index