1980

Using the Purdue PROCSY Terminal System (Pascal)

Bob Brown

Report Number:
80-326

Using the Purdue PROCSY Terminal System

Bob Brown

CSD-TR 326

Purdue University
Computer Science Department
West Lafayette, Indiana

ABSTRACT

This document describes how to use the PROCSY terminal system to create and run Pascal programs. The reader is taught how to log on to a terminal, how to log off, how to create new files, how to send them off for execution, how to modify them, and how to save them between terminal sessions. The material is directed to students in the Introductory Programming course with examples taken from the Pascal language.

January 14, 1980

January 14, 1980

Using the Purdue PROCSY Terminal System

Bob Brown

CSD-TR 326

Purdue University
Computer Science Department
West Lafayette, Indiana

1. INTRODUCTION

This paper gives you the information you'll need to effectively use the Purdue PROCSY terminal system to create and run Pascal programs. Read the document completely before sitting down at a terminal; pay particular attention to the section how to "log off" as it is not good to walk away from a terminal that is "logged on" to the computer under your course account.

This document is organized in several parts just as the computer system is organized in several parts. In using the terminal system, you will be dealing with several programs; each is described separately. For example, there is a program for creating a file, one for changing it, one for running it, and one for saving it.

PROCSY is an acronym, that is, the letters in the name each stand for a word. You will encounter several acronyms when dealing with computers. Here is an explanation of two you'll encounter at Purdue.

PROCSY - Purdue Remote Online Console SYstem. PROCSY is the name of the overall terminal control system including the programs (software) and machines (hardware) required to run the terminals. PROCSY involves several computers besides the main system you've been using. Typically, these additional computers are not of concern to the average PROCSY user.

PIRATE - Purdue Interactive Remote Access Terminal Environment. PIRATE is a program, a part of PROCSY, which you will use. It accepts and interprets commands from terminals to perform most of the functions you'll need to do.

Note: In this paper, the apostrophe character has been used to denote the Pascal quote characters, instead of the more familiar underscore _ or back arrow ←.

2. LOCATING PROCSY TERMINALS

Public PROCSY terminals can be found in several locations. These terminals are subject to advance reservation. A sign-up sheet for the terminals is usually posted one day in advance. Users may write their name down for one hour blocks on a particular terminal. If a signed-up user does not show up within ten minutes after his reserved time, the terminal goes up for grabs. For example, if you want to use a terminal at 2:30 on a Wednesday, it is a good idea to place your name on the sign-up sheet sometime Tuesday morning. Rarely can you show up at a terminal area and find an unreserved terminal for the current or next one hour time block.

The computing center maintains public PROCSY terminals at the following locations:

ENAD - many printing terminals are available and several video terminals.
Math Science - four terminals are available in the Basement. These typically are fully reserved 24 hours in advance.
Stewart Center - a few terminals are available in room G129.

Additionally, in each of the above locations, there is an "express terminal" which allows you to log on for a maximum of ten minutes. These are for "quick fixes" to already written programs.

The Computer Science department has a dozen terminals in rooms MS 433 and MS 435 for students in undergraduate programming courses. These terminals are subject to the same reservation policy stated above. Contention for the terminals in these two rooms varies greatly depending on when courses have projects due but in general, you can get a terminal if you sign up a day in advance.

3. PRELIMINARIES

The keyboard on the terminals is similar to a typewriter in many ways (the keys are mostly all in the same place). However, there are some special keys that are very important. Since there are several different brands of terminals on campus, the description given here may not exactly fit the terminal you use but it should be very close.

KEY	SPECIAL MEANING
RETURN	Use RETURN to denote the end of a line of input to the computer. This key is analogous to the CARRIAGE RETURN button on electric typewriters. The computer will not accept what you type in until you hit this key.
CTRL-H	This is a special key sequence generated by holding down the key marked CTRL (some terminals call it CONTROL) and striking the H key. CTRL-H erases the last character you typed as input. If you make a typing mistake, enter a CTRL-H to erase that character. Multiple CTRL-H's erase multiple characters. On most terminals, the carriage or printing mechanism backs up for each CTRL-H you type.
RUB OUT	This special key, marked DELETE or just DEL on some terminals, causes the computer to throw away the line you were in the process of typing. PROCSY prints the characters ^R to confirm that it ignored the line.
CTRL-B	This is called the "attention" signal. In general, whenever a terminal program is running you can stop it by typing CTRL-B. Also, CTRL-B is used to wake up a terminal when you first come up to it (see LOG ON procedures).
ESC	When you are displaying a long file at your terminal, you can stop it by striking the ESC key. Typing the ESC a second time will cause resumption of the output to your terminal.

4. LOGGING ON

When you first approach the terminal, you must "log on" before PROCSY will let you access the computer. Use the following steps:

1) If the terminal is not already turned on, flip its power switch to the ON position.
2) Check for a switch marked something like ON LINE/OFF LINE. If you find one, make sure it is in the ON LINE position.
3) Strike the CTRL-B attention character. The system should respond with a two line message ended with the question ACCOUNT?
4) Type in your course account number, a comma, and your three letter user-id (e.g. 7711Ø,IJK)
5) The system will respond with the question PASSWORD?
6) Type in your password. Notice that the terminal will not display it as you type so be careful. Hit RETURN as usual when you've finished typing it.
7) The system will respond with several lines of messages. Read these as there may be something interesting printed. In particular, if the system displays something like BY so-and-so YOUR TERMINAL SESSION MUST BE TERMINATED, make note of the time it says because you must finish at the terminal by then or PROCSY will force you off. Eventually, the message SYSTEM? will appear.
8) Type PIRATE and hit the RETURN key or simply strike the RETURN key. These two variations have exactly the same meaning.
9) PROCSY will respond with the confirmation PIRATE and display three plusses +++. This is your verification that PIRATE is in control of your terminal and is ready for a command.

Here is a sample of want you may see go on, the underlined portions are what you would type.

```
TCB L135  12:35:Ø4  11/Ø1/79  FULL DUPLEX
ACCOUNT? 7711Ø,IJK
PASSWORD? (user types his password and RETURN key)
THIS USER LAST LOGGED OFF AT 1Ø:32:ØØ 11/Ø1/79
BY 13:35 YOUR TERMINAL SESSION MUST BE TERMINATED
SYSTEM?  (user types RETURN key)
PIRATE
+++
```

Once you've gotten to this point, PIRATE is waiting for you to type one of its commands. Many of these are described below.

5. USE OF THE TERMINALS FOR PROGRAM DEVELOPMENT

When you were using cards, you punched your entire program onto a single deck. When using the terminal system, you type your entire program into a "program file". A file is simply a sequence of lines stored somewhere internally by the computer. The files you will be using are stored on "disk packs" which are rotating discs with a magnetic coating. The recording technique is very similar to that used in magnetic tapes except that the disk packs look more like a stack of brown phonograph records.

Many of the operations you perform on data files have analogies to operations of card decks. Some of these basic operations are listed below.

Create a file - this means you are entering your program or data into a file for the first time. The analogy to cards is sitting down at the key punch with your program written out and punching a deck.

Edit a file - this means you are making changes to the contents of a file. For example, if a file contains a Pascal program with errors, you must

edit it to correct the errors. The analogy to cards is obvious.

Running a file - this means you instruct the computer to begin the execution of the control cards in the file. These control cards, in turn, may cause a Pascal compilation (a PASCAL card) and a program execution (an LGO card). This is analogous to putting a card deck in a reader and hitting the RESET button.

Saving a file - since most files disappear when you log off the terminal, you must explicitly save files if you expect to be able to use them the next time you log on. If you forget to save a file before logging off, you have effectively thrown it (and the the work required to create it) away.

All files under PROCSY have names. You have probably already been exposed to some named files in previous projects (examples are DATA2, DATA4, etc.). When you create a file, you must choose a name for it. The rules for file names are about the same as those for choosing names for variables in Pascal, except that they may be seven characters long. Valid file names include PROJECT, MYFILE, TEST. Invalid file names include DATA#1, THISFILE, and $SYSTEM.

Each line in a file has a unique line number. Typically, these are real numbers which start at 1.000 and proceed upwards by ones. The contents of the files you need to create will look a lot like the card deck you've been using so far. For example, line 1.000 should be your account number card, and so on. The general form of a data file containing a Pascal program is given below.

```
LINE #    CONTENTS
 1.000    77110,IJK
 2.000    PASCAL
 3.000    PFILES(GET,DATA5,ID=H3B)
 4.000    LIBRARY(PASCLIB)
 5.000    LGO(DATA5)
 6.000    #EOR
 7.000    (*
 8.000     *   YOUR PASCAL PROGRAM GOES HERE
 9.000     *)
           .
           .
           .
46.000    END.
47.000    #EOR
```

A file in this form is a special kind of file called a "program file". Notice some special things about it. First, there is no PASS= line in the file as there is in a card deck. PROCSY does not require you to put it there for security reasons; besides, it already knows your password. Second, notice the lines containing the characters #EOR. Since the terminals do not have multipunch keys, this is what PROCSY uses for the 7-8-9 multipunch you used on cards. Third, notice that the deck is ended with a 7-8-9 multipunch equivalent, not a 6-7-8-9. You could use a #EOF for the 6-7-8-9 but this is not necessary since PROCSY will put it there for you when it is needed.

6. THE PIRATE COMMANDS

PIRATE allows many different commands, though you only need to learn a small handful of them. Typically, each command has two forms: a long form and a short form. This document teaches you the short forms of the commands. It isn't really necessary to learn the long forms and often they are more

confusing. See the references listed at the end of this paper for other documents that tell you all there is to know about these and many other PIRATE commands.

Each of these PIRATE commands run small programs at your terminal to perform some specific function. A handy summary list appears on a separate page at the end of the paper. Some programs are easy to use; others are as complicated as programming itself. The remainder of the body of this manual is dedicated to describing the functioning of the following PIRATE commands:

CREATE - used to create a new data file.
DISPLAY - types the contents of a file on your terminal.
EDIT - used to make changes to the contents of a file.
XMIT - sends a program file off to be run by the computer.
PREVIEW - used to look at how a previously XMITted job ran.
ROUTE - sends the output generated by a job to a line printer.
PUT - saves a file in your personal files storage area (Pfiles).
GET - retrieves a file from Pfiles.
LOG - logs you off the system and ends your terminal session.

The form of all these commands is basically the same. This form is

+++<command>

or

+++<command> <word>

or

+++<command> <word>,<word>, ...

where <command> is one of the described commands and <word> is a filename or something else depending on the particular command. The examples shown below use either blanks or commas to separate parts of each command. However, blanks and commas are treated about the same way by PIRATE; a single comma is the same as one or more blanks. A sequence of blanks can be replaced by a single comma. However, two or more consecutive commas cannot be replaced by any number of blanks and retain the same meaning. For example, all the following commands are equivalent:

+++CREATE,PROJECT
+++CREATE PROJECT
+++CREATE PROJECT

But these are not the same as

+++CREATE,,PROJECT

6.1. The CREATE Command

Use CREATE to build a new file, one that did not previously exist. CREATE prompts you for the lines of the file you plan to create. The form of the CREATE command is

+++CREATE filename

where "filename" is the name of file you want to create.

The prompts used by CREATE are numbers followed by an equal sign. Each

time this prompt appears on your terminal, you can type the next line of your program. Remember the organization of a program file: control cards first, a #EOR, and then your program. After typing each line, be sure to hit the RETURN key so the computer will accept it.

When you've typed in your entire program, enter a line containing the characters #S (short for #STOP) to let CREATE know you're done. When you do this, CREATE gives control of your terminal back to PIRATE who acknowledges it by displaying its +++ prompt. Below is a sample of running the CREATE program.

```
+++CREATE PROJECT
   1.000=77110,IJK
   2.000=PASCAL.
   3.000=PFILES(GET,DATA5,ID=H3B)
   4.000=LIBRARY(PASCLIB)
   5.000=LGO(DATA5)
   6.000=#EOR
   7.000=(*
   8.000= *     PROGRAM TO COMPUTE PERMUTATIONS
   9.000= *)
     .
     .
     .
  46.000=END.
  47.000=#S
+++
```

When you've created your file, it's a good idea to immediately save it since this is easy to forget during your first few times at the terminal. Refer to the PUT command description for details.

An often neglected but useful fact is that PROCSY remembers the name of the file you used the last time you typed CREATE. This is called the "most recent file". Several of the other programs use this "most recent file" in a special shortened form of their commands.

6.2. The DISPLAY Command

DISPLAY does what its name implies; it copies the contents of a file to your terminal. The standard form for DISPLAY is

```
+++DISPLAY filename      (print contents of file named "filename")
```

where "filename" is the name of the file you want to be displayed. The shortened form for the command is

```
+++DISPLAY               (print contents of "most recent file")
```

which displays the contents of the "most recent file". If you use the longer form, the filename you specify becomes the "most recent file" for subsequent shortened commands. The output from DISPLAY is the contents of the file, each line prefixed with its line number. For example, the file PROJECT that was created above would display as

```
+++DISPLAY PROJECT
   1.000=77110,IJK
   2.000=PASCAL
     .
     .
     .
```

```
   46.000=END.
+++
```

There are other interesting forms of the DISPLAY command. Two of these are

```
+++DISPLAY filename,SUP   (print contents of "filename" without line
                           numbers)
+++DISPLAY,,SUP           (display contents of "most recent file" without
                           line numbers)
```

Notice the two commas are needed in the second version to emphasize that the filename was omitted.

6.3. The XMIT Command

The program files you create not automatically executed by the system. Use the XMIT command to cause a program file to be "sent off" for execution. The operation of XMIT is very analogous to placing a program deck into a card reader and hitting the RESET button. Two forms of the XMIT command are

```
+++XMIT,,filename
+++XMIT
```

The first one (note the two commas) sends the program file whose name is whatever you type for "filename" off for execution. The second form sends the "most recent file" off for execution. Every job you send with the XMIT command is given a name. This name will be chosen by XMIT and displayed on your terminal in a message like

```
JOB "Q1" SENT
```

The job name, in this example Q1, is called the "most recent job". If, for some reason, you want to choose your own job name, you can place it in between the commas in the first form or after the XMIT in the short form.

Jobs you send off are placed in a "batch stream". As that job is executing, PIRATE maintains control of your terminal so you can go on with other things. Most often, after sending a job to the batch stream, all you'll do is wait for it to run and look at its output, using the PREVIEW command.

6.4. The PREVIEW Command

Once you've created a batch job by typing the XMIT command, you can check on its progress and look at its output by using PREVIEW. Two forms of the PREVIEW command are

```
+++PREVIEW
+++PREVIEW jobname
```

The first form will run PREVIEW for the "most recent job". The second will run PREVIEW for the jobname specified in the command. This jobname must be one from a previous XMIT command.

If the job you've selected to PREVIEW has not completed running, PREVIEW displays a table of information concerning its current status in the batch stream. This information is updated every eight seconds until the job completes running. Once the job has completed, PREVIEW displays all the output from the job at your terminal. This output will be almost identical to the printout you would normally get automatically if you ran the same job from a card deck. The ESC key (as described above) can be used to suspend the output temporarily until

you type ESC again. CTRL-B aborts the output at your terminal and returns control to PIRATE.

There are other interesting forms of the PREVIEW command. If the job you send off is a Pascal compilation and execution, then

+++PREVIEW,,RØ

displays only the listing of your program, not the dayfile or program output. Also,

+++PREVIEW,,R1

displays only the output generated by your program, not the dayfile or program listing. This latter form is especially useful if you've got all the compilation errors out of your program and are debugging its operation. Similarly,

+++PREVIEW,,D,R1

displays only the dayfile and the program output and skips over the listing of your program. When running on slow printing terminals, these forms can save large chunks of time. Notice, in each of these forms, the two consecutive commas means that the "most recent job" is to be accessed. Place an explicit job name between these commas to specify some job other than the most recent. If you do that, however, the specified job becomes the "most recent job".

6.5. The ROUTE command

Route copies the output generated by an XMITted job to a PUCC lineprinter in ENAD, the Math Science basement, the Stewart Center ground floor, or any other place where a printer exists. The form for ROUTE can get complicated. The general command looks like

+++ROUTE jobname AS bin# TO site

where

bin# is some bin number from Ø to 999. If you leave the AS bin# portion out, ROUTE picks a random number as the bin to use. Luckily, it displays this number on your terminal.

site is some location, preferably the one you're at. Select one of MATH, ENAD, or STEW; each having the obvious interpretations. If you omit the TO site portion out, ROUTE sends your printout to Math Science, an undesirable feature if you are working at ENAD.

When you complete your programming project, use ROUTE to produce a good copy to turn in. Examples of ROUTE commands are

```
+++ROUTE Q1 AS 91Ø TO MATH
+++ROUTE Q7 AS 765 TO STEW
+++ROUTE Q3 AS 555 TO ENAD
+++ROUTE Q3 TO ENAD AS 555
```

The last two examples are equivalent.

6.6. The PUT Command

Files disappear when you log off. PUT copies a file to a place in the computer system where it won't go away when you log off. This place is called

Pfiles (pronounced P-files). Each file that you want to save must be explicitly placed into Pfiles, otherwise it will disappear. The general form of the PUT command is

```
+++PUT filenames
```

as in

```
+++PUT PROJECT
+++PUT PROJ1,PROJ2,MYDATA
+++PUT PROJ1 PROJ2 MYDATA
```

PUT takes a few moments to run so be patient. When PUT has completed copying a file to Pfiles, it will confirm by displaying the name of the file and its size in computer words. A typical PUT command and its result looks like

```
+++PUT TRY1,TRY2
  TRY1     375 WORDS
  TRY2     392 WORDS
```

Files copied to Pfiles do not stay there forever. Any files that you don't touch for fifteen days are automatically removed by the system. Each time you PUT or GET a file, this fifteen day counter is reset. Generally, during the development of a program for a class project, you work on a file every other day or so. Thus, there's little chance that something will be removed from Pfiles unless you're really done with it.

6.7. The GET Command

Just as PUT copies a file to Pfiles, GET copies it back. Thus, whenever you log on to work on a previously CREATEd and PUT file, you must GET it back to use it. The form of the GET command is

```
+++GET filenames
```

as in

```
+++GET PROJECT
+++GET PROJ1,PROJ2,MYDATA
+++GET PROJ1 PROJ2 MYDATA
```

GET takes a few moments, too. When the copy is complete, a confirmation is displayed on your terminal just like PUT does. The last filename specified in the GET command becomes the "most recent file" for subsequent shortened commands. Remember, this is the same as with the CREATE command.

Once you GET a file back, you can make changes to it (see EDIT), run it (XMIT), and resave it (PUT). It is important to note that most of the commands only operate on files that have been copied from Pfiles. Only GET itself operates directly on files in Pfiles.

There exists a special form of GET that your can use to retrieve files from accounts other than your own. This special form is

```
+++GET file//userid
```

Perhaps you want to look at the course supplied data file. An example of how you might do this is

```
+++GET DATA5//H3B
```

The code after the two slashes (the "userid") is the log-on id of the account under which the file was put.

6.8. The LOG Command

Type LOG when you've completed your terminal session. The work you do at the terminal is charged to your class account. If you forget to log off before you leave, anyone can walk up to the terminal and use the computer under your account. This person also would have complete access to all your files, including the capability to look at them, alter them, and delete them! The form of the LOG command is

```
+++LOG                    (log off the computer)
```

PROCSY confirms your log-off with a message which looks something like

```
TCB L265 14.09.25 11/04/79
ESTIMATED SESSION COST $   .04
PLEASE TURN OFF TERMINAL. TNX.
```

It isn't really necessary to turn off the terminal if the next scheduled person is breathing down your back.

6.9. The EDIT Program

EDIT is complicated for the beginner, but he cannot avoid it. Use EDIT to make changes to an existing file. Two basic forms of the EDIT command are

```
+++EDIT filename
+++EDIT
```

Both commands cause EDIT to take control of your terminal and work on a specific file ("filename" in the first case, the "most recent file" in the second). If you use the first form, "filename" becomes the "most recent file".

EDIT has its own set of commands. These are typically one or two letters long, as opposed to the longer word-like commands that PIRATE accepts. EDIT uses the pound sign # as a prompt. Whenever you see +++ as a prompt, you know PIRATE is in control and waiting for a command. Whenever you see a # as a prompt, you know EDIT is in control and waiting for a command.

EDIT makes extensive use of the line numbers in your file. Whenever you refer to a particular line, you can do it by stating its number (e.g. 3P prints line number three).

EDIT accepts several short commands. Some of these are

```
P - Print a line
D - Delete (remove) a line
I - Insert new lines
C - Change parts of a line
R - Replace old lines with new ones
N - Advance to the next line
T - Move to the top of the file
B - Move to the bottom of the file
S - Stop EDIT and return to PIRATE
```

EDIT operates in one of two modes: "command mode" and "insert mode". In

command mode, EDIT prompts with the pound sign and expects one of its commands. In insert mode, EDIT behaves like the CREATE program, accepting new lines from your terminal and placing them into the file. Typing a pound sign by itself in column one while in insert mode causes EDIT to revert to command mode. The I and R commands cause EDIT to switch from command to insert mode.

Many of the commands to EDIT share a common form. This is

#<line><command>

where <line> identifies a line and <command> is any of the one letter EDIT commands. There are several ways to identify a line. One way is to state the number of the line to which you want the command to apply. Remember that you do not need to specify the full number; you can use, say, 34 for line 34.000. Another way is to type a character pattern in between two delimiting characters. Most often, people use slashes for the delimiters. An example of this use applied to the P (print) command is

```
#/pattern/P          (print the next line containing "pattern")
```

as in

```
#/IF ( A/P
```

which will print the next line in your file containing the pattern "IF (A". Two special characters can be used for <line> in the commands. One is the exclamation mark ! which refers to the last line in the file. The other is the up arrow or carat ^ which refers to the first line in the file.

The general scheme for using EDIT is to make changes in your file from the top to the bottom, though this order is not mandatory. Use the commands to make changes and insertions and deletions and then display parts of the file to make sure that what you think should have happened did in fact happen. Many of the commands automatically display the line changed after the change is made. This is useful in assuring yourself that EDIT did what you expected. The key to successful editting is patience and practice. Since you rarely type in a program with CREATE and have it work perfectly the first time, EDIT is unavoidable. Be patient! EDIT is difficult to learn and you're bound to make some mistakes along the way. Think about each command before you hit that RETURN key. Be especially careful in the use of commands that delete lines in your file; once they're gone, they're gone for good.

Each of the EDIT commands is described in a separate section below.

6.9.1. The EDIT I (Insert) Command

The I command causes EDIT to switch to "insert mode". While in insert mode, EDIT prompts with line number in a fashion identical to CREATE (in fact, CREATE is actually EDIT doing a special thing). The general form for I is

#<line>I

where <line> has the usual interpretation. Examples are

```
#I                  (insert after the current line)
#12I                (insert after line 12)
#/END/I             (insert after the next line containing "END")
#!I                 (insert at the bottom of the file)
```

The first form causes new lines to be inserted after the current line in the

file. The second form causes lines to be inserted after number 12; these new lines are numbered as 12.Ø1Ø, 12.Ø2Ø, and so on. The third form causes lines to be inserted after the next line in the file containing the string END (note that this could be END, SENDER : INTEGER, or WRITELN(' FRIENDLY COMPUTER')). The fourth form inserts lines at the bottom of the file.

If you insert between two currently existing lines, EDIT chooses line numbers with decimal portions, as in 1.Ø1Ø, 43.15Ø, 23.ØØ1, and so on. The ordering of the lines in your file is sequential based on the line numbers. An example of the use of Insert is

```
#8I
  8.Ø1Ø=    WRITELN(' X, Y = ', X, Y );
  8.Ø2Ø=    WRITELN(' ------------------------');
  8.Ø3Ø=#
#
```

Some older documents teach the I command as the INSERT command. These two are equivalent; I is easier to type than INSERT.

6.9.2. The EDIT P (print) Command

Use the P command to display lines in your file. One form is

```
#<line>P
```

which causes the line referred to by <line> to be printed. In this case, <line> becomes the current line. If <line> is omitted, the current line is used. Another form is

```
#<line>P<target>
```

where <target> can take on one of three forms. If you type <target> as an unsigned integer, the resulting P command will print that many lines starting at <line>, as in 4P7 (which displays 7 lines starting at number 4). If you type <target> as a poundsign followed by an integer constant, P prints from <line> to the line number specified in <target> as in 4P#7 (which prints from line 4 to 7). If you type <target> as a pattern delimited by (enclosed in) special characters, P prints from <line> through the next one containing the pattern. Thus, the following are all valid forms for the P command:

```
#P                  (print the current line)
#12.Ø2P             (print line 12.Ø2)
#1ØP4               (from line 1Ø, print 4 lines)
#/IF/P/END/         (print from the next line containing IF to the next
                     containing END)
#P#32.Ø2            (print down to line number 32.Ø2Ø)
#^P!
```

The last form prints all lines in your file (from the first ^ to the last !).

6.9.3. The EDIT D (delete) Command

Lines can be deleted or removed from a file by using the D command. To delete a single line, the form is

```
#<line>D            (delete line <line>)
```

as in examples

```
#D                    (delete the current line)
#27D                  (delete line 27)
#/END/D               (delete the next line containing the pattern "END")
```

To delete a range of lines, the form is

```
#<line>D<target>
```

as in examples

```
#D2                   (delete the next two lines)
#2.1D#8               (delete lines 2.100 through 8.000)
#45D/END/             (delete from line 45 through the next one  containing the pattern "END")
```

The D command sets the current line to the next line in the file immediately after the last one that was deleted. If you delete the last line in the file, EDIT types the message BOTTOM OF FILE REACHED.

6.9.4. The EDIT R (replace) Command

The R command operates like a combination of the D and I commands. It deletes one or more lines and then enters insert mode so you can enter new lines. One form is

```
#<line>R              (replace line <line>)
```

This form deletes line <line> and enters insert mode. New lines are given numbers that fall between the preceeding line and the next line. Example of this form include

```
#R                    (replace the current line)
#12R                  (replace line 12)
#:A/B:R               (replace the next line containing the pattern "A/B")
```

Another form is

```
#<line>R<target>
```

which allows you to replace more than just one line. Examples of this form are

```
#R2                   (replace the next two lines)
#R#5                  (replace lines from the current one down to line number 5)
#13.5R/IF/            (replace from line 13.500 through the next one containing the pattern "IF")
```

You can replace a line or lines by any number of new lines. Once EDIT enters insert mode, you type in new lines as necessary. Remember that typing a pound sign # in column one causes EDIT to revert to the normal command mode.

An example of the operation of the Replace command is:

```
#8R
  7.010=   WRITELN(' ', X, Y, ZETA );
  7.020=   IF X > Y THEN
  7.030=      BEGIN
  7.040=      WRITELN(' X IS GREATER THAN Y');
  7.050=      X := X - 1.0
```

```
7.Ø6Ø=       END;
7.Ø7Ø=#
#
```

6.9.5. The EDIT N (next line) Command

The N command has but one purpose: to change the current line. The most commonly used forms are

#N	(advance to the next line)
#<line>N	(advance to the line after the one specified by <line>)
#N<target>	(advance <target> (plus or minus) lines)

Simple examples include

#N2	(skip the next two lines)
#N-3	(go back three lines)

The functioning of the N command can be performed by simply stating the line number you want to become the current line, as in

#<line>	(advance to the line specified by <line>)

Examples of the previous form include

#27	(make 27.ØØØ the current line)
#/WHILE/	(make the next line containing WHILE the current line)
#+4	(skip down four lines)
#-1	(backup one line)

The N command by itself performs no changes on your file. Rather, it moves the current line around so that some of the other commands (D or I or R or C) become simpler in form.

6.9.6. The EDIT T (top) Command

The T command makes the first line in your file the current line. The form is simply

#T	(move current line to the top)

6.9.7. The EDIT B (bottom) Command

The B command makes the last line in your file the current line. The form is simply

#B	(move current line to the bottom)

If you are working on a file that you previously copied from Pfiles with the PROCSY GET command, there will be an end-of-record mark #EOR at the bottom of your file whether you put it there or not. Thus, typing the B command will make that #EOR line the current line. If you then want to add lines with the I command, it is best to Replace that #EOR instead of using I directly. Otherwise, the #EOR will remain above the new lines you insert.

6.9.8. The EDIT MO (move) Command

If you need to move one or more lines in your file from one place to another, use the MO command. The most common form of the command is

```
#<line>MO<target> <range>
```

The command moves the lines starting at <line> through <range> to immediately after <target>. Both <target> and <range> must be preceeded by a pound sign (#) if they refer to line numbers, otherwise they refer to line counts. Examples are:

```
#MO #20.01 6        (move 6 lines starting at the current line to
                     immediately after line number 20.01)
#5MO #45 #10        (move lines 5 through 10 to immediately after
                     number 45)
#MO 5               (move the current line down five lines)
#MO /STOP/          (move the current line down to follow the next con-
                     taining the pattern "STOP")
```

Another variation of the MO command is the CO command. CO (copy) has the same form as MO. Instead of moving lines, it copies them. That is, it leaves them where they are and makes a copy of them at the specified <target>.

6.9.9. The EDIT C (change) Command

Use the C command to change portions of existing lines. Very often the types of programming mistakes you make can be corrected without replacing entire lines of your file. All you really need to do is correct one little part of an existing line. This is what C does. The most basic form is

```
#C/oldstring/newstring/
```

where "oldstring" is the sequence of characters in the existing line that you want to replace. The C command scans the line for "oldstring". If it is found, it is replaced by "newstring". Also, "oldstring" and "newstring" do not need to be the same length. Examples are

```
#C/O/P/             (change the first occurrence of O in the current
                     line to the letter P)
#C/IF/IF (/         (change the pattern "IF" in the current line to "IF
                     (")
```

These forms operate only on the current line. Another form is

```
#<line>C/old/new/
```

which operates just like the first form but on the line specified by <line>, which then becomes the current line. Another form is

```
#<line>C/old/new/ <target>
```

as in

```
#C/ALPHA/BETA/ #23  (change the first occurrence of "ALPHA" to "BETA"
                     in all lines from the current one down to number
                     23.000)
#C;X / Y;X * Y; 3   (change the first occurrence of "X / Y" in the next
                     three lines)
```

A special form involving an asterisk exists for the C command. This is

```
#<line>C/old/new/ <target> *
```

This complicated command changes all occurrences of the pattern "old" to the pattern "new" from the line specified by <line> down to and including the line specified by <target>. In this form, <line> may be omitted to refer to the current line. The asterisk * signifies that all occurrences of "old" on a single line are to be replaced, as opposed to just the first occurrence on each line. The most common example of the use of this extended C command is

```
#^C/old/new/ 1 *
```

which changes all occurrences of the string "old" in your file to "new". If applied to a file containing a Pascal program, the command

```
#^C/GAMMA/COUNT/ 1 *
```

might be used to rename all occurrences of GAMMA (which might be a variable name) to COUNT.

This C command is probably the most powerful of all the EDIT commands and must be learned well before you can use the editor competently. Study this section again. Try the C command out on some files you create.

Here is an example of the use of the Change command.

```
#5P13
   5.000=    WHILE FOND OR I <= M DO
   6.000=        IF LIST(I) = PATTRN THAN
   7.000=        BEGIN
   8.000=        FOUND = TRUE;
   9.000=        WHERE := I;
  10.000=    ELSE
  11.000=        I := I + 2;
  11.010=    IF FOUND THEN
  11.020=        WRITELN(' SUCCESSFUL');
  11.030=    ELSE
  11.040=    WRITELN('UNSUCCESSFUL');
#5C/FOND/NOT FOUND/
   5.000=    WHILE NOT FOUND OR I <= M DO
#C/OR/AND/
   5.000=    WHILE NOT FOUND AND I <= M DO
#C/M DO/N ) DO/
   5.000=    WHILE NOT FOUND AND I <= N ) DO
#C/ I / ( I /
   5.000=    WHILE NOT FOUND AND ( I <= N ) DO
#N
   6.000=        IF LIST(I) = PATTRN THAN
#C/(I)/[I]/
   6.000=        IF LIST[I] = PATTRN THAN
#C/THAN/THEN/
   6.000=        IF LIST[I] = PATTRN THEN
#9I
   9.010=        END
   9.020=;
   9.010=        END
#11C/2/1/
  11.000=        I := I + 1;
#0C/=/:=/
```

```
   8.000=          FOUND := TRUE;
#/UNSUCC/C/'U/' U/
  11.040=     WRITELN(' UNSUCCESSFUL');
#C//    /
  11.040=          WRITELN(' UNSUCCESSFUL');
#-2C/;//
  11.020=          WRITELN(' SUCCESSFUL')
#5C/FOUND/DONE/ #11.04 *
   5.000=     WHILE NOT DONE AND ( I <= N ) DO
   8.000=          DONE := TRUE;
  11.010=     IF DONE THEN
```

6.9.10. The EDIT S (stop) Command

The S command causes EDIT to return control of your terminal to PIRATE. The form is simply

```
#S                    (return to PIRATE)
```

as in

```
#S
+++
```

An alternative to the S command is the LOG command. Typing LOG to EDIT causes it to return to PIRATE.

6.10. Other PIRATE Commands

There are many other commands accepted by PIRATE; some are useful and some are not. A few of these are described below.

The FILES command lists all the files that are currently available at your terminal. Type

```
+++FILES              (list all files available at your terminal)
```

The INDEX command lists all the files that you currently have stored away in Pfiles. Remember that those files are not directly accessible at your terminal. They must be copied from Pfiles with the GET command. Type

```
+++INDEX              (list all files stored in Pfiles)
```

The HELP command is very useful. If you forget how to use one of the other PIRATE commands, type

```
+++HELP command       (show how to use "command")
```

HELP prints a bunch of information about how to use the command whose name you type after the word HELP. To see all the command names the HELP knows about, type

```
+++HELP               (show list of PIRATE commands)
```

There are many commands that PIRATE accepts that are generally useless to the beginner so don't waste too much time typing HELP for all the commands listed.

The Help command has abbreviated forms. One of these is

+++EXAMPLES command (show examples of a command)

EXAMPLES shows only examples of the use of a command, not the full explanation. Often this is all that is needed to jog the memory.

Another useful command is

+++SYNTAX command (show short command syntax)

SYNTAX displays one line showing the general syntax of the command and what it expects as arguments.

To rename a file in your Pfiles area, use

+++RENAME oldfile,newfile

Renaming a file in Pfiles does not change the name of any copies you may have at your terminal (made by a GET command). To rename a file that you've already copied from Pfiles, use

+++QRE oldfile,newfile (rename a local file)

Note that this second form doesn't change the name of the copy in Pfiles.

The DELETE command removes a file or files from your Pfiles storage area. The form is

+++DELETE filenames (remove files from your Pfiles storage area)

Examples include

+++DELETE OLDFILE (remove the file named OLDFILE from your Pfiles storage area)

+++DELETE PROJ1,PROJ2

Use the DELETE command judiciously. Don't delete files for projects unless you absolutely have to.

If you're interested in seeing how much money you've spent on your class account, type

+++PFILES LIST,CHARGES

7. A Sample Program in a File

The following is an example of what a Pascal program in an existing PROCSY file might look like, complete with control cards.

```
+++DISPLAY PROG
  1.000=77110,IJK
  2.000=PASCAL
  3.000=PFILES(GET,DATAF,ID=H3B)
  4.000=LIBRARY(PASCLIB)
  5.000=LGO(DATAF)
  6.000=/EOR
  7.000=PROGRAM SOLVE(INPUT,OUTPUT);
  8.000=(*
  9.000= *   PROGRAM TO SOLVE QUADRATIC EQUATIONS
 10.000= *
 11.000= * INPUT :
 11.000= *   A, B, C : COEFFICIENTS
 12.000= *
 13.000= * OUTPUT :
 14.000= *     X1, X2 : ROOTS OF THE EQUATION
 15.000= *)
 16.000=VAR
 17.000=    A, B, C : REAL;
 18.000=    X1, X2 : REAL;
 19.000=    DISC : REAL;
 20.000=(*
 21.000= *   READ INPUTS...
 22.000= *)
 23.000=BEGIN
 24.000=READ ( A, B, C );
 25.000=(*
 26.000= *  SOLVE IF POSSIBLE"
 27.000= *)
 28.000=DISC = B * B - 4.0 * A * C;
 29.000=IF DISC < 0.0 THEN
 30.000=    WRITELN(' ROOTS ARE IMAGINARY')
 31.000=ELSE
 32.000=    BEGIN
 33.000=    DISC := SQRT ( DISC );
 34.000=    X1 := ( -B + DISC ) / ( 2.0 * A);
 35.000=    X2 := ( -B - DISC ) / ( 2.0 * A );
 36.000=    WRITELN(' ROOTS EXIST...', X1, X2)
 37.000=    END
 38.000=END.
 39.000=/EOR
+++
```

8. Summary of PIRATE Commands

Listed below are several of the common forms of the PIRATE commands described above.

```
+++CREATE filename          (Accept lines from the terminal and insert them
                             into the file named "filename")

+++DISPLAY                  (display contents of "most recent file")
+++DISPLAY filename         (display contents of "filename")
+++DISPLAY,,SUP             (display "most recent file)
+++DISPLAY,filename,SUP

+++XMIT                     (send "most recent file" off for execution)
+++XMIT,,filename

+++PREVIEW                  (display output of "most recent job")
+++PREVIEW,jobname
+++PREVIEW,,R1              (displays output only)

+++ROUTE jobname            (goes to MATH with a random bin #)
+++ROUTE jobname TO site AS bin#

+++GET filenames            (copy "filenames" from Pfiles)
+++GET filename//userid

+++PUT filenames            (copy files into Pfiles)

+++EDIT                     (edit most recent file)
+++EDIT filename            (edit "filename")

+++LOG                      (log off PROCSY)
```

9. Summary of EDIT Commands

Here is a summary of the more common forms of the EDIT commands described above.

```
#I                          (insert after current line)
#<line>I                    (insert after line identified by <line>)

#P                          (print current line)
#<line>P                    (print line identified by <line>)
#<line>P<target>            (print a range of lines)

#D                          (delete the current line)
#<line>D                    (delete line identified by <line>)
#<line>D<target>            (delete a range of lines)

#R                          (replace the current line)
#<line>R                    (replace the line identified by <line>)
#<line>R<target>            (replace a range of lines)

#N                          (skip to next line)
#N<target>                  (skip to line identified by <target>)

#<line>MO<target> <range>
#<line>CO<target> <range>
```

#C/old/new/	(change a pattern of characters in the current line)
#<line>C/old/new/	
#<line>C/old/new/ <target>	
#<line>C/old/new/ <target> *	

1Ø. References

Reference material for the topics covered in this paper are available in limited supply in the Math Science Building room B-6. PUCC (Purdue University Computing Center) documents are typically not tutorials; rather, they are reference material. The following documents may prove handy to one interested in bettering his skills in using PROCSY and EDIT beyond the scope of this paper.

LØ-PROCSY describes all the features of PROCSY that you'd ever want to know. This document does not describe PIRATE or EDIT. What else is there? (you may ask). Basically, all that's left is how to log on.

LØ-PIRATE describes in general terms the operation of some of the PIRATE commands. The material duplicates the material in this document but takes a different approach.

LØ-PIRMAC describes the functioning of the more interesting PIRATE programs (PREVIEW, CREATE, DISPLAY, etc.). It is a general reference document describing all the PIRATE "macros" (the PIRATE term for a program).

LØ-QED is the reference document for EDIT. QED is the given name for EDIT; EDIT is a PIRATE program (macro) that simply executes QED. All the EDIT commands explained above and more are described in this paper.

LØ-PIRREF is not recommended for the beginner. It describes all the PIRATE "primitive" commands. Primitives are basically commands that aren't macros (get it?). They are all three letters long and most of them begin with the letter Q (as in QRE, QDE, QSI, etc.). Some are useful, many you'll never figure out.

THE TERMINAL PROCESS

This flowchart shows the basic steps and commands normally used when working on a program at a terminal.