17329105 D

CONTROL DATA CORPORATION

SOFTWARE TOOLS
USER'S MANUAL

CONTROL DATA®
MP-32
COMPUTER SYSTEMS

17329105

CONTROL DATA
CORPORATION

MP-60 COMPUTER SYSTEM

Software Tools
User's Manual

REVISION RECORD

REVISION	DESCRIPTION
A 01-19-82	Original Release.
B 05-25-82	Incorporate changed Ratfor Coding Conventions and the "cdata" statement.
C 06-08-82	Re-order sections and numbering.
D 12-27-82	Change section numbering and make miscelaneous editorial changes.

1 MP-60 ED - Text Editor

1.1 INTRODUCTION

Ed is a text editor. Ed operates on a copy of any file it is editing; changes made in the copy have no effect on the file until a w (write) command is given.

Commands to ed have a simple and regular structure: zero, one, or two line addresses followed by a single character command, possibly followed by parameters to the command. The structure is:

[line],[line]command <parameters>

The '[line]' specifies a line number or address in the buffer. Every command which requires addresses has default addresses, so the addresses can often be omitted.

Line addresses may be formed from the following components:

17	an integer number
.	the current line
$	the last line in the buffer
.+n	"n" lines past the current line
.-n	"n" lines before the current line
/pattern/	a forward context search
\pattern\	a backward context search

Line numbers may be separated by commas or semicolons; a semicolon sets the current line to the previous address before the next address is interpreted. This feature can be used to determine the starting line for forward and backward context searches ("/" and "\").

1.2 REGULAR EXPRESSIONS

Ed supports a form of regular expression notation for specifying patterns in line addresses and in the s command. A regular expression specifies a set of strings of characters. That is, regular expressions can be created which can be used to locate or change patterns only at particular positions on a line, patterns which contain certain characters plus perhaps others, or that match text of indefinite length. Regular expressions are constructed as

follows:

1. An ordinary character (not one of those discussed below) is a regular expression and matches that character.

2. A percent "%" at the beginning of a regular expression matches the empty string at the beginning of a line.

3. A dollar sign "$" at the end of a regular expression matches the null character at the end of a line.

4. A question mark "?" matches any character except a newline character.

5. A regular expression followed by an asterisk "*" matches any number of adjacent occurrences (including zero) of the regular expression it follows.

6. A string of characters enclosed in square brackets "[ ]" matches any character in the string but no others. If, however, the first character of the string is an carret "^" the regular expression matches any character except the characters in the string (and the newline).

7. The concatenation of regular expressions is a regular expression which matches the concatenation of the strings matched by the components of the regular expression.

8. The null regular expression standing alone is equivalent to the last regular expression encountered.

9. One to three integers delimited by underscores "_" within a regular expression "tabbing" of the search to the column indicated.

If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be escaped by preceding it with an atsign "@".

For example, the regular expression

```
/%a?*b/
```

would search for a line which began with "a" and had a "b" in it, possibly with other characters between "a" and "b".

The empty regular expression // causes the last regular expression given to be used.

1.3 COMMANDS

Following is a list of ed commands. Default addresses are shown in parentheses:

(.)a
<text>
.

The append command reads the given text and appends it after the addressed line. '.' is left on the last line input, if there were any, otherwise at the addressed line.

b filename

The batch command submits a file for batch execution. If the file currently being edited is to be submitted, it first must be written with the w command.

(.,.)c
<text>
.

The change command deletes the addressed lines, then accepts input text which replaces these lines. '.' is left at the last line input, if there were any, otherwise at the first line not deleted.

(.,.)d

The delete command deletes the addressed lines from the buffer. The line originally AFTER the last line deleted becomes the current line; however, if the lines deleted were originally at the end, the new last line becomes the current line. Repeatable with '-'.

e filename

The edit command causes the entire contents of the buffer to be deleted and then the named file to be read in. '.' is set to the last line of the buffer. The number of lines read is typed. 'filename' is remembered for possible use as a default file name in a subsequent r or w command.

f filename

If 'filename' isn't given, the command prints the currently remembered file name. If 'filename' is given, the currently remembered file name is changed to 'filename'.

(1,$)g/regular expression/command

In the global command, the given command is executed for every line which matches the given regular expression.

h

The help command causes a list of ed commands to be displayed.

(.)i
<text>
.

The insert command inserts the given text BEFORE the addressed line. '.' is left at the last line input, or if there were none, at the addressed line. This command differs from the "a" command only in the placement of text.

(.,.)k<address>

The kopy command copies the addressed lines to the position after the line specified by <address>. The last of the copied lines becomes the current line. Repeatable with '-'.

l<coll,coln>

The length command changes the portion of the text lines which is displayed. Coll is the beginning column, and coln is the ending column. The initial default is columns 1 through 80.

(.,.)m<address>

The move command repositions the addressed lines after the line specified by <address>. The last of the moved lines becomes the current line. Repeatable with '-'.

o seqnum

The output command retrieves an output file from the MPX/OS output queue. Seqno specifies the file desired. The file is copied into ed's buffer and released. The remembered filename is not changed.

(.,.)p

The print command prints the addressed lines. '.' is left at the last line printed. The p command may be placed on the same line after any other command to cause printing of the last line affected by the command. Repeatable with '-'.

q

The quit command causes ed to exit.

(.)r filename

The read command reads in the given file after the addressed line. If no file name is given, the remembered file name is used (see e and f commands). The remembered file name is not changed. Address '0' is legal for this command and causes the file to be read

in at the beginning of the buffer. If the read is successful, the number of lines read is typed. '.' is left at the last line read in from the file. Repeatable with '-'.

(.,.)s/regular expression/replacement/ or,
(.,.)s/regular expression/replacement/g
The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, the first occurrence of the expression is replaced by the replacement specified. If the global replacement indicator g appears after the command, all occurrences of the regular expression are replaced. Any character other than space or newline may be used instead of the slash '/' to delimit the regular expression and replacement. A question mark '?' is printed if the substitution fails on all addressed lines. '.' is left at the last line substituted. If the regular expression is not specified (e.g. s//pat/), the last regular expression given is used.

An ampersand '&' appearing in the replacement is replaced by the string matching the regular expression. (The special meaning of '&' in this context may be suppressed by preceding it by '@'.)

Lines may be split or merged by using the symbol '@n' to stand for the newline character at the end of a line.

This command is repeatable with '-'.

t
t<coll,...,coln>
t<char>
The tab command controls ed's tabulation feature. Tab alone displays tabulation status. Tab with col parameters sets or resets the tab columns. Tab with a non-numeric character resets the tab character (default is ;). "Tab 0" disables tabulation, which is the default.

u
The upper command toggles the processing of input characters. If upper is ON, lower case input characters are converted to upper case. Default is OFF.

(1,$)w filename
The write command writes the addressed lines onto the given file. If the file does not exist, it is created. The remembered file name is not changed. If no file

name is given, the remembered file name is used (see the e and f commands). '.' is left unchanged. If the command is successful, the number of lines written is typed.

(1,$)x/regular expression/command
The except command is the same as the global command except that the command is executed for every line except those matching the regular expression.

za filename
The zap command deletes the given file. The remembered file name is not changed. If no file name is given, the remembered file name is used.

zo<coll,coln>
The zone command determines the operational column range used by the pattern search commands. Text that lies outside of the range specified by zone is not considered for pattern matching. The default zone is columns 1 thru 136.

(.)=
The line number of the addressed line is typed. '.' is left unchanged.

(.+1)<carriage return>
An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to '.+1' and thus is useful for stepping through text.

-
A minus alone repeats the previous repeatable command. Repeatable commands are those marked as such above.

1.4 DIAGNOSTICS

The error message "?." is printed whenever an edit command fails or is not understood.

1.5 SUMMARY OF SPECIAL CHARACTERS

The following are special characters used by the editor:

Character	Usage
?	Matches any character (except newline)
%	Indicates beginning of line
$	Indicates end of line or end of file
[...]	Character class (any one of these characters)
[^...]	Negated Character class (any character except these)
*	Closure (zero or more occurences of previous pattern)
@	Escaped character (e.g. @%, @[, @*)
&	Ditto, i.e. whatever was matched
c1-c2	Range of characters between c1 and c2
@n	Specifies the newline mark at the end of a line
_nn_	Specifies "go to column" in a pattern search

1.6 COMMAND SUMMARY

Addresses:

17	a decimal number
.	the "current" line
$	the last line of the file
/pat/	search forward for line containing pat
\pat\	search backward for line containing pat
line+n	n lines forward from line
line-n	n lines backward from line

Defaults:

(.)	use current line
(.+1)	use the next line
(.,.)	use current line for both line numbers
(1,$)	use all lines

Commands:

(.)	a	append text after line (text follows)
	b file	submit file for batch execution
(.,.)	c	change text (text follows)
(.,.)	d	delete text

	e file	discard current text, enter file, remember file name
	f	print file name
	f file	remember file name
	h	print help information
(.)	i	insert text before line (text follows)
(.,.)	k line3	copy text to after line3
	l col,col	set line display window
(.,.)	m line3	move text to after line3
	o seq	retrieve output file from queue
(.,.)	p	print text (can be appended to other commands)
	q	quit
(.)	r file	read file, appending after line
(.,.)	s/pat/new/g	substitute new for leftmost pat (g implies all occurrences)
	t	display tab columns
	t c	set tab character
	t col,..,col	set tab columns
	u	toggle uppercase forcing
(1,$)	w file	write file, leave current text unaltered (if "file" isn't given, write to current filename)
	za file	delete file
	zo col,col	set search zone
(.)	=	print line number of current line
(.+1)	<CR>	print line
	-	repeat last command
(1,$)	g/pat/command	do command on lines containing pat (except a, c, i, q commands)
(1,$)	x/pat/command	do command on lines not containing pat (except a, c, i, q commands)

1.7 REFERENCES

The "Ed User's Guide" section of this manual.
"A Tutorial Introduction to the ED Text Editor"
by B. W. Kernighan
Kernighan and Plauger's "Software Tools", pages 163-217
The Unix command "ed" in the Unix manual.
"Edit is for Beginners" by David A. Mosher (available from
UC Berkeley Computer Science Library)
"Edit: A Tutorial" (also available from the
UC Berkeley Computer Science Library)

1.8 DIFFERENCES FROM OTHER EDS

This editor differs from the one originally provided in Kernighan's Software Tools in the following ways:

Both upper and lower case commands are recognized.

The "b", "h", "k", "l", "o", "t", "u", "za", "zo", and "-" commands have been included.

The "_nn_" construct has been included in the regular expression definition.

There are several discrepancies between this editor and Unix's ed. These include:

1. Unix uses 'v' instead of 'x' for the except command.

2. Unix uses a caret (^) instead of percent (%) for the beginning-of-line character.

3. Unix uses '.' instead of '?' to indicate a match of any character.

4. Unix uses backslash (\) instead of atsign (@) for the escape character.

5. Unix uses a question mark (?) instead of a backslash (\) to delimit a backward search pattern.

6. The Unix 'r' command uses the last line of the file, instead of the current line, as the default address.

7. The Unix editor prints the number of characters, rather than lines read or written when dealing with files.

THIS PAGE INTENTIONALLY BLANK

2 MP-60 RATional FORtran - Structured Fortran Preprocessor

2.1 INTRODUCTION

RATional FORtran is a preprocessor which implements several extensions to ANSI Fortran. It was conceived by Kernighan and Plauger for their book "Software Tools" (Addison-Wesley, 1976). All of the example code in the book is written in Ratfor, including a sort program, an editor, a text formatter, and Ratfor itself.

Ratfor may be used as a Program Design Language (PDL) by ignoring the Fortran output and including, as Ratfor comments, those parts of the design not detailed enough to specify in Ratfor statements. Conventions for use of Ratfor as a PDL are found in a later section.

The major advantage of Ratfor when used to produce Fortran code is that it doesn't obscure the program with the syntactic peculiarities of Fortran, yet it insures that the program translates readily into Fortran.

The Ratfor which has been implemented for the MP-32 includes some minor extensions that implement listing control; otherwise it is identical to Kernighan's Ratfor.

2.2 SUMMARY

Here is a summary of Ratfor. Essentially, Ratfor IS Fortran, except that several new statements have been added to make it easier to specify flow of control. At the same time, Ratfor provides some notational conveniences, so programs can be made more readable.

In the following paragraphs, a "statement" is any legal Fortran statement or any Ratfor statement. Any Ratfor or Fortran statement or group of statements can be enclosed in braces "{" and "}" to make a compound statement, which is then equivalent to a single statement, and usable anywhere a single statement can be used.

The "if" statement:

```
if (condition)
        statement 1
else
        statement 2
```

If "condition" is true, do "statement 1", otherwise do "statement 2". The "else" part is optional. Ambiguity is removed by the following rule: "An else is paired with the last un-elsed if."

The "while" statement:

```
while (condition)
        statement
```

If "condition" is true, "statement" is executed until "condition" is false.

The "for" statement:

```
for (initialize; condition; reinitialize)
        statement
```

This is equivalent to:

```
{
initialize
while (condition){
        statement
        reinitialize
        }
}
```

If "condition" is omitted, it is taken to be always true. If either "initialize" or "reinitialize" is omitted, the corresponding part of the Fortran expansion is omitted. A "next", discussed below, causes control to be transferred to "reinitalize", not "condition" however.

The "repeat-until" statement:

```
repeat
        statement
        until (condition)
```

The "statement" is done one or more times until "condition" becomes true, at which time the loop is exited. The "until" part is optional; if it is omitted, the result is an infinite loop, which must be broken some other way.

The "break" statement:

```
break
```

The "break" statement causes whatever loop it is contained in to be exited immediately. Control resumes with the next statement after the loop. Only one loop is terminated by a "break", even if inside several nested loops.

The "next" statement:

```
next
```

causes whatever loop it is contained in to go immediately to the next iteration, skipping the rest of the loop body. "next" goes to the "condition" part of a "while", "do" or "until", to the top of a "repeat" loop, and to the "reinitialize" part of a "for".

The "do" statement:

```
do limits
        statement
```

sets up a standard Fortran "do" loop. "limits" must be a legal Fortran "do" specification, like i=1,n.

The "cdata" statement:

```
cdata array "upper and lower case text"
```

causes the generation of a Fortran Data statement. The data placed in the statement are decimal numbers coresponding to the ASCII translations of the text given. The symbol EOS must be defined before the cdata statement is encountered. The value of EOS is placed in the array after the last character of the text string. The array would normally be a

character array. The cdata statement is an MP-32 extension of Kernighan's Ratfor.

The "define" statement:

```
define(symbol,replacement)
```

causes occurances of "symbol" to be replaced by "replacement" in the expanded output.

By using "define" one can create mnemonic symbols which can then be used throughout the Ratfor program. These symbols contain only letters and digits; once defined, they may be used anywhere, surrounded by non-alphanumerics (except within hollerith constants delimited by quote marks).

Example: An array size may be defined as "define(BLARGSIZ,1000)" then "dimension(BLARGSIZ)" is legal, as well as "do i=1,BLARGSIZ".

By convention, the "symbol" part of the definition is capitalized.

The "other" statement:

```
other
```

may be any legal Fortran statement or, in fact, any junk that you like. The "other" statement is really anything that Ratfor cannot decode into one of the previous Ratfor statements. Ratfor passes these statements through to the Fortran input file and Fortran will further decode them.

Continuation. Generally, the end of a line marks the end of a statement, but constructions like:

```
if (c == NEWLINE)
      nl = nl + 1
```

are obviously not finished after the line that contains the "if" and so they are continued automatically. This is also true of conditions that extend over more than one line, as in:

```
while (c == BLANK
       | c == TAB
       | c == NEWLINE)
             i = i + 1
```

Lines ending with a comma are also continued. Continuation may be forced by terminating the line with an underscore character "_". The underscore is not printed.

Hollerith constants. Character data surrounded by quote marks is changed to Fortran hollerith representation. For example:

```
900 format ("this is a test")
```

would become

```
900 FORMAT(14HTHIS IS A TEST)
```

Symbol substitution does not take place within the character data.

Relationals. Ratfor uses several short-forms for the Fortran relational operators ".xx.". These are:

==	.EQ.	<	.LT.	>	.GT.	&	.AND.	^	.NOT.
^=	.NE.	<=	.LE.	>=	.GE.	|	.OR.		

Comments. A sharp sign # anywhere in a line signals the beginning of a comment, which is terminated by the end of the line.

Listing Control. Ratfor as implemented on the MP-32 has been extended to include listing control. All of the following controls must begin in column 1 of the input line. The controling characters will not be displayed in the listing.

To indicate the start of a new module:

```
#M modulename - module description
```

should be included. This construction will cause a new page to be started and trap the modulename so that it may be included in the listing header.

To turn the full listing on or off:

```
#N          Suppress the listing (No list)
#L          Enable the listing (List)
```

To turn Common Module listing on or off:

#CN	Suppress Common module listing
#CL	Enable Common Module listing

To indicate the start of a Common Module:

```
#CMmodulename - description
```

should be included. This construction is listed, even when Common Module listing is suppressed.

To indicate the end of a Common Module:

```
#CE
```

should be included. This terminates suppression of the list output when Common Module listing is suppressed.

Control Statement. The MP-60 Control Statement which invokes Ratfor is as follows:

```
*RATFOR(I=inlun,L=listlun,E=errlun,F=ftnlun,A,R)
```

inlun	Logical unit containing Ratfor input lines. Default is 63. If I is standing alone, default is 56.
listlun	Logical unit to receive Ratfor listing or 0. Default is 62. If L is standing alone, default is 60.
errlun	Logical unit to receive Ratfor error diagnostics. Default is 62. If E is standing alone, default is 60.
ftnlun	Logical unit to receive Fortran statements. Default is 59. If F is standing alone, default is 56.
A	If A is present, errors cause the job to abort after all input has been processed.
R	Produce "readable" fortran output.

BNF. Below the Ratfor language is presented in Backus-Naur Form.

```
program   |statement
          |program statement
statement|IF( condition ) statement
          |IF( condition ) statement ELSE statement
          |WHILE( condition ) statement
          |FOR( initialize; condition; reinitalize ) statement
          |REPEAT statement
          |REPEAT statement UNTIL( condition )
          |DO limits statement
          |digits statement
          |BREAK
          |NEXT
          |{ program }
          |OTHER
```

2.3 CODING CONVENTIONS

Ratfor does not enforce any coding conventions. What is included here is, by and large, a codification of the coding conventions obviously in use for the SOFTWARE TOOLS book. Therefore, the overall style, format, and level of in-code documentation, if not specified below is to be obtained from the example code in SOFTWARE TOOLS, Kernighan & Plauger; Addison-Wesley, 1976.

1. All code and documentation use the full set of 95 printable ASCII characters.

2. The only non-comment characters in the Ratfor code that may be upper case are those strings that constitute Ratfor Definition invocations. I.E. define(BIGB,$42) causes all occurances of "BIGB" in the Ratfor source to become "$42" before presentation to Fortran. By convention, symbols created by define statements are upper case and are therefore involked with upper case.

3. All modules begin with a single line which has a number sign (#) in column 1, followed by a "M", a single blank, and the module name. Following the name are a single blank, a minus sign, a single blank, and a single sentence describing the function of the module. See Listing Control.

4. The indenting interval is 3 spaces. Listing controls are the only constructions allowed in columns 1 to 3.

5. Stand alone comments should start on the indenting level to which they refer and comment the code below them. Comments included on a code line should start in column 40 if the code permits.

6. Opening braces should be included on the line that contains the controlling statement. Example:

```
if (i == 2400){        # if clock = midnight
```

7. Closing braces should stand on a separate line at the indenting level of the compound statement that they close. Example:

```
j = 25 + k
m = BIGC
}
```

8. Common Modules consist of:

   a. #CMmodulename - description

   b. Single lines which type each variable in the block:

```
integer a       # Count of alphas, init=0
integer b       # Index to bees, init=1
integer c(25)   # Array of bees
```

   c. One blank line.

   d. The "common" declaration, e.g.

```
common /blk/ a,b,c
```

   e. #CE common-module-end listing control

9. Only one statement is allowed per line, except for the "else if" construct.

The above coding conventions are employed in the following Ratfor coding example. A narative of the example is given following the example.

```
/COMDECK CLOOK                                                          01
#CMclook - Define common data structure for token lookup.               02
   integer lastp             # last used in namptr; init=0              03
   integer lastt             # last used in table; init=0               04
   integer namptr(MAXPTR)    # name pointers                            05
   character table(MAXTBL)   # text of names and defns                  06
                                                                        07
   common /clook/ lastp, lastt, namptr, table                           08
#CE                                                                     09
/DECK DEFINES                                                           10
#M defines - Definitions for Sample Routine                             11
   define(MAXDEF,200)        # Max char in a definition                 12
   define(MAXTOK,200)        # Max characters in a token                13
   define(MAXPTR,1500)       # Max defines in a lookup                  14
   define(MAXTBL,15000)      # Max characters in all defines            15
   define(EOS,$FD)           # End-of-String character                  16
   define(NO,0)              # Function response                        17
   define(YES,1)             # Function response                        18
/DECK LOOKUP                                                            19
#M lookup - Locate name and extract definition from table.              20
   integer function lookup(name, defn)                                  21
                                                                        22
   integer lookup            # Returned =YES if name was                23
                             # found or =NO if not found.               24
   character name(MAXTOK)    # Name to lookup.                          25
   character defn(MAXDEF)    # Definition as copied from table.         26
                                                                        27
   # The routine "lookup" indexes into the "table" array by             28
   # using the pointers stored in the "namptr" array.  At each          29
   # pointed location in "table" an attempt is made to match            30
   # the string given in "name".  If a match is found, the              31
   # definition string in "table" is moved to the "defn"                32
   # parameter.                                                         33
                                                                        34
   integer i                 # Local index variable                     35
   integer j                 # Local index variable                     36
   integer k                 # Local index variable                     37
   character ch(50)          # Character String                         38
   integer chw(1)               # ditto                                 39
   integer s2                # Index to the blarg                       40
                                                                        41
   CALLS scopy, others,                                                 42
         more, calls, could, go, here                                   43
   CALLSR funcr                                                         44
                                                                        45
   ESRS writlu, ust, ctoi,                                              46
        more, esrs, could, go, here                                     47
                                                                        48
/CALL CLOOK                                                             49
                                                                        50
   equivalence (ch, chw)                                                51
                                                                        52
```

```
cdata ch "Preset the text"                                          53
                                                                    54
data s2 /4092/          # Preset the variable                       55
                                                                    56
                                                                    57
for (i=lastp; i>0; i=i-1){                                          58
   j = namptr(i)                                                    59
   for (k=1; name(k)==table(j) & name(k)^= EOS; k=k+1)               60
      j = j + 1                                                     61
   if (name(k)== table(j)){        # got one                       62
      call scopy(table, j+1, defn, 1)                               63
      lookup = YES                                                  64
      return                                                        65
      }                                                             66
   # This is an example of the "else if" construct.                67
   else if (1 == 3){                                                68
      s2 = s2 + 1                                                   69
      }                                                             70
   }                                                                71
lookup = NO                                                         72
return                                                              73
end                                                                 74
```

The following is a narative of the example given above:

01-01 This code will reside in an Update Oldpl. See Update Manual.

02-02 "#CM" indicates "Common Module". See Listing Controls.

03-08 Common definitions. See Common Module conventions.

09-09 "#CE" indicates end of "Common Module".

10-10 Update directive.

11-11 Module Title line.

12-18 Ratfor Define Statements. Comments begin in column 30.

19-19 Update directive.

20-20 Title line. "#M " causes page eject places "name" in Ratfor header until next "#M ". The blank in "#M " is significant and "name" ends at the first blank encountered.

21-21 Program, function, or subroutine statement.

22-22 One blank line.

23-26 Parameter declarations. Comments start in column 30.

27-27 One blank line.

28-33 Function text section.

34-34 One blank line.

35-40 All local variables are declared in this section. Comments start in column 30. If variables are arrays, the array dimensions must be declared in this section.

42-43 All routines called are declared here. The Define CALLS is translated into "integer" before the statement is passed to Fortran.

44-44 Function "funcr" is to return a REAL value. The Define CALLSR translates into "real" for Fortran. Other Defines are:

CALLSD - double precision
CALLSL - logical

45-45 One blank line.

46-47 Declare ESRS used by the routine. The Define ESRS is translated to "external" for Fortran.

48-48 One blank line.

49-49 Update Commondeck Call Directives for common data.

50-50 One blank line.

51-51 Equivalence statements. See local declarations section for use of the comment "ditto".

52-52 One blank line.

53-53 Cdata statements. Cdata is translated into a data statement by Ratfor. Characters are translated into decimal numbers. Upper/lower case is preserved. The array to be filled must be declared in the local variables section.

54-54 One blank line.

55-55 Data statements. Comments start in column 30.

56-57 Two blank lines.

58-74 Executable portion of the program. Comments start in column 40.

2.4 RATFOR AS A PROGRAM DESIGN LANGUAGE

Ratfor may be used as a program design language at several levels of the program design task.

While the initial design is being conceived and hiearchy charts produced, Ratfor can be used as a repository for information pertaining to each module of the design. As such, Ratfor comments will describe the function of the module and define its parameters.

Later, Ratfor Common Modules can be used to describe the data structures as they are defined. Each module which will use a particular piece of common data can have an UPDATE /CALL statement included at this time.

Finally, as the internal designs of each module are created, a mixture of true Ratfor statements and special comments will describe the design of each module. Special comments are those which describe sections of the design which have not yet been detailed into Ratfor code; these comments take the following form:

```
#< comment
```

Thus the #< sequence is unique in the text and may be extracted in order to find all those sections of a program not yet complete.

Obviously, when the design has been detailed into Ratfor statements, the Fortran output of Ratfor may be compiled and program testing begun.

The above sequence allows all design documentation to be carried through the project in an orderly way, and no design documentation is produced which is not immediately available to the detailed designer, implementor, tester, or maintainer of the module/program.

It should be noted that since this design process is a continuum, individual projects should determine points in the design sequence which are appropriate for peer or

external review; thus maintaining an open design process.

3 SYSDUMP - SYStem panic DUMP interpreter

3.1 Synopsis

```
]j sysdump
Seq = xx   Output = PR

]f ocf
OCF.

accept,xx,jmgr

]f
SYSDUMP

12/09/81      RESIDENT=01   LIBRARY=00    SYSABRT=27D9

LOWMEM =0000  PDUMP  =1265  ESRS   =1374  IOMGR  =170D  IOITS  =1A5C
ERMGR  =2065  EVNMGR =2232  INTRCOM=23EC  TASKMGR=24AC  DAYMGR =2890
DAYDMP =2B1C  SYSQS  =2CB4  FILEMGR=32A4  SYSBIO =3B0C  STDIO  =3E64
ITS    =402E  ICF    =420F  LOF    =498E  OCF    =4B9C  FCF    =510F
DMF    =5373  DCF    =54E6  BCF    =55C4  SMDMGR =56C2  FDDMGR =59B7
CRTMGR =5B43  BCLAMGR=5C40  UCLAMGR=5E32  CRDMGR =6011  MT05MGR=619E
LPTMGR =6297  CCCMGRT=6348  RBTMGR =650E  OPFMGR =6AFA  CTT    =6D03
SYSTINT=6D53  MEMPOOL=6E3D

ENTER 1 IF PAGE DUMP DESIRED
      2 IF PROGRAM DUMP DESIRED
      4 IF Z80 MEMORY DUMP WANTED
        CARRIAGE RETURN TO EXIT
```

3.2 Description

SYSDUMP is a tool designed to aid the system analyst in determining the cause of a system abort (panic dump). When a system abort occurs, an image of main memory on the MP-32 may be obtained by loading a magnetic tape on tape unit 0, then readying the tape once it has reached load point. SYSDUMP may then be used to to read the tape and display and interpret all or part of that memory.

Using the abort tape as input, SYSDUMP initially reads in the portion of memory where the Operating System resides and determines and displays: the date abort occured, Resident and Library editions, the address from which the abort was initiated and the addresses of all the routines within the Operating System. It then displays a menu of the options available to the use and awaits a response. (An example of this display appears in Synopsis). Options may be requested in any order and in any quantity.

Once the user has completed selection of whatever combination of options were desired and has exited SYSDUMP, a listing of the results generated by those options will be printed on the line printer.

SYSDUMP provides several different ways of displaying the contents of main memory.

1. Page Dump: Option selected when a 1 is entered in response to menu. This option will display the contents of the registers for each of the 8 Machine States and any 4K page(s) of memory specified.

   A display will be output requesting the page number(s) of the 4K page(s) the user wishes to have listed. For each page requested, the contents will be displayed in both binary and ASCII format, along with additional information to delineate the absolute locations being displayed. The display requesting the page numbers will be repeated after each interaction.

   An individual page is specified by entering the page number followed by a carriage return.

   Contiguous pages may be specified by entering the starting page number, a '-', and then the ending page number, (e.g. 8-A), followed by a carriage return.

   To exit the option, enter a line consisting solely of a carriage return. The general menu will then be repeated, should the user wish to select additional options.

2. Program Dump: Option selected when a 2 is entered in response to menu. This option is used to display specific routines within the Operating System.

   A display will be output requesting the program number(s) of the Operating System routine(s) to be listed, followed by a display of the routine names and their corresponding numbers. For each routine requested, the memory occupied by the routine will be displayed in both binary and ASCII format, along with additional information to delineate both the absolute address within memory, its position relative to the starting location of the routine, and the routine name. The display requesting the program numbers will be repeated after each interaction.

   An individual program is specified by entering the program number followed by a carriage return. If 0 (SPECIAL) is specified, the starting and ending

address of the memory to be displayed will be requested.

Contiguous programs may be specified by entering the starting program number, a '-', and then the ending program number, (e.g. 1-4), followed by a carriage return.

To exit the option, enter a line consisting solely of a carriage return. The general menu will then be repeated, should the user wish to select additional options.

3. Dump Analysis: Option selected when a 3 is entered in response to menu. This option is used to produce an analysis of the state of the system when it aborted.

   Variables and tables used by the Operating System will be listed, threads will be checked. All jobs and their associated tasks will be analyzed and the internal tables associated with them will be listed, and as each is analyzed the job name will be displayed to the user. When analysis is complete, the general menu will be displayed, should the user wish to select additional options.

4. Z80 Dump: Option selected when a 4 is entered in response to menu. This option is used to display Z80 memory.

   The Z80 dump menu is displayed. The particular Z80 to display memory from is the entered. The range of the memory to be dumped can also be specified. Additional memory ranges can be requested and different Z80s dumped.

   To exit the option, enter a line consisting solely of a carriage return. The general menu will then be repeated, should the user wish to select additional options.

3.3 Files

SYSDUMP-FILE - file containing data read in from dump tape.

3.4 References

ITS Version 2 Reference Manual (17329140)

4 TDUMP - Tape/File Dump Utility

4.1 Synopsis

*TDUMP(p1,p2,...pn)

I Input file or tape containing data to be dumped.
- Default: 5
- Standing Alone: 5
- Range: 1-60

L Listing file where tdump output is written.
- Default: 62
- Standing Alone: 62
- Range: 1-62

FB First Block in each file to be dumped.
- Default: 1
- Standing Alone: 1
- Range: 1-n

NB Number of Blocks in each file to dump.
- Default: ALL
- Standing Alone: ALL
- Range: 1-n

BS Maximum Block Size.
- Default: 480
- Standing Alone: 480
- Range: 1-4096

For Disk files BS must be set to the actual block size used for the file.

NL Maximum number of lines to print.
- Default: ALL
- Standing Alone: ALL
- Range: 1-n

NF Maximum number of files to dump.
- Default: 1
- Standing Alone: To Double End-of-file
- Range: 1-n

NR Inhibit initial rewind of input file.
- Default: REWIND
- Standing Alone: NO REWIND

80 Format for 80 column device.
- Default: 120 Column Dump
- Standing Alone: 80 Column Dump

Example:

```
*TDUMP(I=10,BS=512,80,NR,NF=3)
```

4.2 Description

TDUMP is a tool which is used to display the actual contents of a file or a tape. It has various parameters to control the amount and format of the information produced.

A typical usage of TDUMP would be to investigate the content of a file written by a program under checkout to determine if the file was being formatted correctly.

4.3 Files

TDUMP reads a single input file and produces a single listing file. There are no scratch files employed.

5 COPYCF - Copy Coded Files

5.1 Synopsis

*COPYCF(lun1,lun2,n,fchar,lchar,fline,lline)

lun1 Input logical unit for file to be copied; if this parameter is absent, 63 (input) is assumed.

If the input file is 63, an end-of-file is simulated by a line consisting of "*END" in columns 1 to 4.

lun2 Output logical unit for file being generated; if this parameter is absent, 62 (output) is assumed.

n Number of files (decimal) to copy; if this parameter is absent, n=1 is assumed.

fchar First character position of each line to copy; if omitted, fchar=1 is assumed.

lchar Last character position of each line to copy; if this parameter is omitted, lchar=136 is assumed.

fline First logical line from which the copy will begin; if absent, fline=1 is assumed.

lline Last logical line to copy; if omitted, lline=infinite is assumed.

Example:

```
*COPYCF(10,11,,1,80,5,200)
```

This example copies columns 1 to 80 of lines 5 through 200 from logical unit 10 to logical unit 11.

5.2 Description

COPYCF copies ASCII coded files. Parameters are accepted which will control the copy.

If lun1=lun2, n files are skipped but no transfer occurs.

Statistics are produced on LUN 63 when the copy is complete.

THIS PAGE INTENTIONALLY BLANK

6 COPYSCF - Copy Shifted Coded Files

6.1 Synopsis

*COPYSCF(lun1,lun2,n)

lun1 Input logical unit for file to be copied; if this parameter is absent, 63 (input) is assumed.

If the input file is 63, an end-of-file is simulated by a line consisting of "*END" in columns 1 to 4.

lun2 Output logical unit for file being generated; if this parameter is absent, 62 (output) is assumed.

n Number of files (decimal) to copy; if this parameter is absent, n=1 is assumed.

Example:

```
*COPYSCF(10,11,2)
```

This example copies two files from logical unit 10 to logical unit 11.

6.2 Description

COPYSCF copies ASCII coded files in preparation for printing. All input lines are shifted right one character position and a blank is placed in column 1. Also a single line, consisting of a "1" in column 1 is written before each file is copied.

If lun1=lun2, n files are skipped but no transfer occurs.

Statistics are produced on LUN 63 when the copy is complete.

THIS PAGE INTENTIONALLY BLANK

Appendix A - ED User's Guide

A.1 INTRODUCTION

The MP-60 editing program is an extension of the editor described in Brian Kernigan's book SOFTWARE TOOLS (1976 ADDISON-WESLEY) and is known as ed. It is a very fast interactive, line-oriented text editor capable of creating and modifying textual information, computer programs, character data, documents, etc. It has the ability to do line searches with text patterns, make global changes (where global refers to all characters in a line and/or all lines in an entire file) and do simple file I/O operations.

Be aware that this is not a conversational editor but more of a programmer's editor. The commands it uses are very streamlined and terse; it is not very 'chatty' and will do very little 'hand holding' (if an error is made, only a '?' and a beep will be issued).

A.1.1 Entering Ed

Assuming that the user has already logged on, ed may be initiated typing ']J', carriage return (a ']J' is a command to the ITS part of the operating system to submit a non-hard-copy job). If there is not enough memory to run ed, then the message 'JOB NOT INITIATED' is displayed (the ']J' can be tried later when memory becomes available again). If the memory necessary to run ed is available, then the ed job is assigned a sequence number and the message 'SEQ=XX OUTPUT=MS' is displayed. After the message 'Ed Version YY/MM/DD.' is displayed, ed commands can be entered. The]PORT ITS command can be used to show whether the user has an ed or not and whether it is connected or not. If ed is not connected (it does not have an asterisk ('*') in front of the word ED), then the command ']L', carriage return can be used to reconnect back to ed if the user is in the interactive control facility (ICF).

Ed can also be run from batch using the *ED control statement.

*ED(<inlun,>outlun,?erlun,'filespec',-)

inlun Logical unit containing ed commands (input lines). Default lun is 05.

outlun	Logical unit to receive ed outputs (that output which would normally go to the screen). If > is replaced by >>, then outlun is not rewound, and outputs are appended. Default lun is 06.
erlun	Logical unit to receive ed errors. If ? is replaced by ??, then erlun is not rewound, and errors are appended. Default lun is 06.
'filespec'	If present, ed will execute an "edit" command using this file before executing any commands from inlun.
-	If present, linecount outputs for e, r, and w commands are supressed.

A.1.2 Leaving Ed -- Quit

To exit from ed, the user just enters: 'Q', carriage return. Note that leaving ed does not automatically write the working file back to the disk. This must be done with a WRITE command (see section A.2.4 WRITING OUT THE FILE TO MASS STORAGE for more information) before leaving.

A.1.3 Help Command

If there are any questions on the format or the commands that are available, the HELP command can be used. When H(HELP) carriage return is typed in, a list similar to that shown in section A.1.4 is displayed. See section A.1.4 for further explanation of the list.

A.1.4 Command Summary

The Summary below shows most of the commands that are available to ed (the text locating commands are described in section 4.3 TEXT LINE LOCATING). The File Related commands (BATCH, EDIT, FILENAME, OUTPUT, READ, WRITE and ZAP) deal with the control and related operations of disk files. Ed Environment commands (HELP, LENGTH, QUIT, TAB, UPPER, ZONE and =) define or display the working environment of ed. The Text Line Manipulation commands (APPEND, CHANGE, DELETE, INSERT, KOPY and MOVE) are used to handle complete text lines. And the individual text line command (SUBSTITUTE) is used to operate on separate characters of a text line. Section A.3 COMMANDS AND COMMAND FORMAT has a more detailed description of the format of an ed command.

	FORMAT	DESCRIPTION
n	Append	Add lines after n.
	Batch fn,dc	Submit batch job fn.
m,n	Change	Replace lines m thru n.
- g m,n	Delete p	Delete lines m thru n.
	Edit fn	Edit test file fn.
	Filename fn	Define a new filename.
n	Insert	Insert lines before n.
- g m,n	Kopy o p	Copy lines m thru n to o.
	Length l,r	Set margins to l,r.
- g m,n	Move o p	Move lines m thru n to o.
	Output v	Obtain output file v.
- g m,n	Print	Display lines m thru n.
	Quit	Terminate session.
- g n	Read fn	Read file fn after n.
- g m,n	Substitute /p1/p2/g p	Change pattern.
	Tab c	Set tab chatacter to c.
	Tab n1,n2,...,n10	Set column tabs.
	Upper	Fold to uppercase.
g m,n	Write fn	Write m thru n to file fn.
	ZAp fn	Delete file fn.
	ZOne a,b	Set a,b search columns.
g n	=	Display line number.
	-	Repeat last saved ed command.

Where:

a	Starting search column position
b	Ending search column position

c	TAB character
dc	Disposition code
fn	Information necessary to address a disk file. It has the following format: pfn = 1-14 character permanent file name owner = 1-4 character owner name edition = 1-2 character edition access key = 1-4 character permission code
g	Global prefix
l	Starting display column position
m	Ending line designator
n	Starting line designator
n1,n2,...n10	Tab column positions (up to 10)
p	Print postfix
p1	Target pattern
p2	Replacement string
r	Ending display column position
v	Batch job sequence number

A.2 FILES AND FILE COMMANDS

There are two types of disk files available: binary files and ASCII files. Binary files cannot be edited and if retrieval is attempted, an error will be issued. Text files are stored in BLOCKER/DEBLOCKER format and arranged in 480 word blocks.

When an existing text file is requested for editing, each text line is brought in individually, its length is then

determined (it can be up to 160 characters long), a line number assigned and then written out to a working copy of the text file. Since ed operates only on the working copy of the text file the text file is never updated except by an explicit WRITE command (see section A.2.4 WRITING OUT THE FILE TO MASS STORAGE for more information).

When a file is read in or created, each line of a text file is numbered sequentially starting from the beginning of the file. However, these line numbers are internal to ed and are not physically part of the text line. Each time a line is added, deleted, moved or copied, all the text lines are renumbered. If text lines 2, 3, 4 are deleted, then text line 5 becomes the new text line 2, text line 6 becomes the new text line 3, etc. The first text line is always line one and a 0 (zero) may be specified with some ed commands to indicate the text line before the first actual text line.

Ed always keeps in memory the last text line accessed or modified and this is known as the current text line. If there is no last line, for instance when starting out with no text file, then any command (except the text inputting commands A(APPEND) OR I(INSERT)) which specifies the current text line is illegal. The current text line can be specified in line designators to define a range for an ed command to process (see section A.3.1 LINE DESIGNATORS for more information).

The file command format is a simplified version of a text line command. Only the filename (see section A.2.1 SPECIFYING THE FILENAME) is needed and if the command is a READ (see section A.2.3 RETRIEVING AN EXISTING FILE) or a WRITE (see section A.2.4 WRITING OUT THE FILE TO MASS STORAGE), up to 2 line designators may be included.

A.2.1 Specifying the Filename

The filename provides unique identification of a particular text file. When a filename is entered, the information is saved away in the file name buffer. This buffer then can be used to provide information for parts of the filename that are omitted by the user. The filename buffer is updated every time a filename is specified and the filename buffer can be displayed by entering a 'F' (FILENAME command), carriage return. The FILENAME command can also be used to update the filename information by entering 'F' followed by

the filename.

There are four parts to the filename, each separated by a comma: the name of the file, the name of the owner, the edition, and the access key. The name of the file is a 1 to 14 character string which must be included if there is none in the filename buffer. The owner is a 1 to 4 character string which if omitted, will be supplied by the filename buffer. The edition is a 1 or 2 character string which if omitted, is either the edition number in the filename buffer or a '01'. The access key is a 1 to 4 character password string which if omitted is the same as the owner name.

EXAMPLE	MEANING
F	Display the filename buffer.
F FILEABC	Change the name of the file in the filename buffer to FILEABC.
F PLOT,,XY	Change the name of the file and the edition in the filename buffer to PLOT and XY, respectively.

A.2.2 Creating a new file

In order to create a new text file, the filename must first be specified (see section A.2.1 SPECIFYING THE FILENAME for more information). If there is already a working text file, there will be residual text lines which can be deleted (see section A.4.4.3 DELETE) before the new text lines can be entered. To enter text lines use either the INSERT command (see section A.4.4.4 INSERT) or the APPEND command (see section A.4.4.1 APPEND).

A.2.3 Retrieving an existing file

If a MP32 text file already exists, it can be brought in for editing (replacing the old text file) by using the EDIT command. This command uses no line designators (see section A.3.1 LINE DESIGNATORS for more information) and only the filename is needed.

An existing text file can also be inserted anywhere in the current text file by using the READ command. It requires only a line designator (if none is given, then the text file will be read in immediately after the current text line) and the filename.

Immediately after the file is brought in, the number of text lines read in is displayed. If the file does not exist, the error message "FILE MANAGER ERROR 20" is displayed. If there is an attempt to bring in a binary file, the number of lines brought in is zero and a BLOCKER/DEBLOCKER error is issued.

EXAMPLE	MEANING
E TEXT,GES	Bring in the text file named TEXT, the owner being GES, the edition being 01 (if no previous edition) or the previous edition and the access key being GES.
E	Use the previously entered filename to bring in the file for editing (i.e. bring in the same file for editing).
E ,JCS	Bring in the text file specified by the filename buffer except that the owner is JCS and the access key is JCS.
5R FTP,WMY,01,GBF	Append the text file FTP, owner WMY, edition 01, access key of GBF after text line 5 of the current text file.
R JLJ-TEXT	Append the text file JLJ-TEXT after the current text line of the text file.

A.2.4 Writing the file to mass storage

After a text file has been modified, it must be written in order to save the altered file. The current text file or any portion of it can be written back on to the same file or to a completely different file on the disk. Line designators are optional and if not specified, the entire file is written. If one line designator is specified, then that one designated text line is written. If two line designators are given, then the part of the file defined by the first line designator to the second line designator is written. The filename (see section A.2.1 SPECIFYING THE FILENAME for more information), if given, specifies what file to write to. If none is given, then the filename in the filename buffer is used (if there is none, an error is issued). When ed is finished writing out the text file, the number of text lines is displayed.

EXAMPLE	MEANING
W	Write out the whole text file to the disk file specified by the filename in the file name buffer.
W DATA,AHA	Write out the whole text file to the disk file named DATA, owner AHA, edition 01 (or the current text filename's edition number), access key AHA.
1,10W TEMP,RFG,01,DBK	Write out the first 10 lines of the current text file to the disk file named TEMP, owner RFG, edition 01, access key DBK.

A.2.5 Removing an existing file

Ed has the provision to permanently remove an ASCII or binary file by using the ZAP command. The format of the command is ZAP, followed by the filename (see 2.1.0 SPECIFYING THE FILENAME for more information). If the filename is not specified, then the default filename in the filename buffer (which is normally the current text file name) will be used.

Using the ZAP command affects only the permanent file copy. The working text file is still intact and can be operated upon as normal. If the file does not exist, the error 'FILE MANAGER ERROR 20' will be displayed.

EXAMPLE	MEANING
ZAP	Remove the file specified in the file name buffer.
ZAP SOURCE,DWD	Remove the file named SOURCE, owner DWD edition 01, access key DWD.

A.2.6 Executing a job

A file containing control cards can be submitted for batched execution by using the BATCH command. The format of the command is BATCH, followed by the filename (see section A.2.1 SPECIFYING THE FILENAME for more information), followed by the disposition code. If the filename is not specified, then the default filename in the filename buffer (which is normally the current text file name) will be used. If the disposition code is not specified, PR will be used.

The BATCH command only works on the permanent file copy. If the command succeeds, the message SEQ = ss OUTPUT = dc appears. If the file does not exist, the error 'FILE MANAGER ERROR 20' will be displayed.

EXAMPLE	MEANING
BATCH	Submit for batched execution the file specified in the filename buffer.
BATCH JOB,ETA	Submit for batched execution the filename JOB, owner ETA, edition 01, access key ETA.

A.2.7 Retrieving job output saved on mass storage

When a job is submitted for batched execution, the output may be routed to various output devices. It can be automatically placed on the output queue for eventual listing or on a disk file for later inspection. When the job is submitted for execution, a sequence number and an output disposition code is given. The disposition code must be MS for the file output to be retrived by ed (see the]B command in the ITS manual for more info). To retrieve the output file, an 'O'(Output) followed by the sequence number is entered. If the output file does not exist, the message 'FILE MANAGER ERROR 20' will be displayed. If the job is still running when the OUTPUT command is entered, ed will wait until the job is completed and then display the number of text lines when the file is brought in. Note that when the output file is brought in, the disk copy is removed (i.e. the OUTPUT command can be used only once for a particular sequence number). In order to save the output file, it must be written out (see section A.2.4 WRITING OUT THE FILE TO MASS STORAGE).

SEQUENCE	MEANING
B JOB,DBK,01,PGS,MS	Submit the job JOB1, owner DBK, edition 01, access key PGS for execution and put the job output on a disk file.
SEQ = 1A OUTPUT = MS	The output file is given the sequence number 1A.
O 1A	Ed is requested to retrieve the output file 1A (which is the job listing for JOB1).

A.3 COMMANDS AND COMMAND FORMAT

The text oriented commands (for file oriented commands, see section A.2 FILES AND FILE COMMANDS) are simple and easy to use once the command format is understood. A summary of commands is given in Section A.1.4 and can be composed of either upper and/or lower case characters. Blanks that are imbedded in a command line are ignored (except when imbedded in a pattern) and only enough characters that uniquely discriminate an ed command need be entered (the capital letters of the commands shown in section A.1.4 COMMAND SUMMARY are the minumum needed). All commands must have a terminating carriage return, and if a command is entered in error, ed will respond with a '?' and beep.

Usually ed is quite responsive, but occasionally a command will be typed in and nothing will appear to be happening. This is quite normal and is because of ed's double buffering of the input (i.e. it reads another command before it has completed the execution of the current command). Eventually ed will return and be ready for another command.

A command is normally comprised of 3 parts and all 3 parts are optional: the line designators which describe the starting and ending text lines; the command itself; and postfixes which are used to specify post command options for certain ed commands. If there are no line designators, then only the current text line is processed. If only one line designator is given, then that line designator is used to locate to that text line (thus making that text line the current text line) before it is processed. If 2 line designators are given, then the first line designator defines the starting text line and the second line designator, the ending text line. If the command itself is omitted, then the default is to display the last located text line. If the Global postfix is omitted, then the Substitute command will change only the first occurence of a pattern match (see section A.5 INDIVIDUAL TEXT LINE COMMAND -- SUBSTITUTE for more information). If the Print postfix is omitted, then the current text line will not be displayed after the command has processed it. Along with this, a command maybe preceded by a prefix known as a GLOBAL prefix. This prefix has its own line designators and is used to preselect text lines for the accompanying command to process.

The general format for an ed command is as follows:

SLDG ELDG GPRE SLDC ELDC C GPOST PPOST CR

Where: SLDG is the starting line designator for the Global prefix.

ELDG is the ending line designator for the Global prefix.

GPRE is the Global prefix.

SLDC is the starting line designator for the command.

ELDC is the ending line designator for the command.

C is the command.

GPOST is the Global postfix if the command is a Substitute.

PPOST is the Print postfix.

CR is the carriage return key.

A.3.1 Line Designators

Line designators are simple expressions which will resolve into a legal line number and are used to specify a range or to locate to a specific line (see section A.2 FILES AND FILE COMMANDS). The expression can be comprised of integer numbers, representative line number symbols, pattern string operands and plus/minus sign operators. The operands and operators can be in any order, respectively.

A.3.1.1 Line Numbers

Any non-negative integer number can be used in the line designator.

LINE DESIGNATOR	RESOLVED LINE NUMBER
5	5
6+1	7
3-1+2	4

A.3.1.2 Line Number Symbols

Ed keeps track of two text line numbers at all times: the current text line number and the line number of the last text line.

The current text line number is represented by a period character ('.') and the last line number is represented by a dollar sign ('$'). Both of these characters can be used in the line designators.

LINE DESIGNATOR	RESOLVED LINE NUMBER
$-4	If the last line number was 11, then the resolved line number is 7. The line designator can also be interpreted as requesting the 4th line from the bottom of the file.
.+3	If the current line number is 6, then the resolved line number is 9. This line designator is effectively asking for the 3rd text line after the current line.
$-.+1	Somewhat unusual, but assuming that the current line number is 2, and the last line number is 7, then the resolved line number is 7-2+1 = 6.

A.3.1.3 String Patterns

String patterns are descriptions of a set of characters which can be used to identify or locate a particular text line. They are enclosed by a pair of delimiters (a delimiter is any character not found in the pattern which is used to define the start and end of the pattern) and the delimiter may or may not have an implied search direction associated with it. A text pattern can be a simple thing, like the letter 'a', or a more elaborate pattern built up from simple elements. To build any desired pattern, only a few simple rules need be remembered. Any literal character, like 'q' is a text pattern that matches the same character in the text line being scanned. A sequence of literal characters, like 'ABC' is a pattern that matches any occurence of that sequence of characters in the text line.

A pattern is said to match part of a text line if the text line contains an occurence of the pattern. For example, the pattern 'AA' matches the line 'AABC' once at column position 1, the line 'AABCAABC' in 2 places (column positions 1 and 5), and the line 'AAAAAA' in 5 (overlapping) places. Matching is done on a text line by text line basis; no pattern can be matched beyond the end of the line.

Although it is possible to describe a pattern using nothing but a literal string, ed has provisions to describe more general patterns using pattern elements. Pattern elements are comprised of meta characters which are reserved characters or symbols which have special meaning when used in a pattern.

ESCAPE ('@') Meta Character

The ESCAPE meta chracter says to ed: "Treat the next character following the '@' as a literal character and not as a meta character (that is, turn off the special meaning of the character)".

PATTERN	MEANING
@?	Treat the ANY-CHARACTER meta character ('?') as a literal '?'.
@@	Treat the second '@' character as a literal '@'.
@x	Treat the literal chracter 'x' as a literal 'x'.

BEGINNING-OF-LINE ('%') Meta Character

When the '%' meta character is placed at the beginning of the pattern, it informs ed to try to match the pattern starting only at the beginning of the text line. If the '%' appears by itself in a pattern, then it is specifying the beginning of the text line (see section A.5 SUBSTITUTE command for more information and examples). If the '%' does not appear as the first or only character of a pattern, then it is treated as a literal character.

PATTERN	MEANING
%ABC	The pattern will be matched only if a 'ABC' occurs in the first 3 characters of a text line.

PATTERN	MEANING
A%BC	The pattern will be matched if a 'A%BC' is found anywhere in the text line.
%	Request the beginning of the text line.

END-OF-LINE ('$') Meta Character

When the '$' meta character is placed at the end of the pattern, it informs ed to try to match the pattern only if the pattern occurs at the end of the text line. If the '$' appears by itself in a pattern, then it is specifying the end of the text line (see section A.5 SUBSTITUTE command for more information and examples). If the '$' does not appear as the last or only character of a pattern, it is treated as a literal character.

PATTERN	MEANING
ABC$	The pattern will be matched only if 'ABC' occurs in the last 3 characters of the text line.
$ABC	The pattern will be matched if a '$ABC' is found anywhere in the text line.
$	Request the end of the text line.

ANY-CHARACTER ('?') Meta Character

The ANY-CHARACTER meta character is a place holder character in a string. When ed encounters a '?' in the pattern, and there is a corresponding legal character in the text line, then the '?' would match that character. That is, the ANY-CHARACTER would match any legal character.

PATTERN	MEANING
A?C	The pattern will be matched if 3 characters are found such that the 1st character is a 'A', the 3rd character is a 'C' and the 2nd character is any character (a match would be found for 'ABC', 'AXC', 'A C', 'A4C', etc.).

COLUMN-POSITION ('_') Meta Character

The COLUMN-POSITION meta character is used to specify a particular column position to match the remaining portion of the pattern. To describe a column position in a pattern, the column number must be between 1 and 136 and be enclosed before and after with an underscore ('_').

When used in a SUBSTITUTE command, the COLUMN-POSITION meta character is able to locate to any column position (even though the position maybe beyond the end of the text line) and continue matching of the remaining portion of the pattern (see section A.5 SUBSTITUTE command for more information and examples). This allows insertion of strings anywhere in or beyond the end of a text line.

PATTERN	MEANING
/_5_/	The COLUMN-POSITION meta character is being used in a text line search and is requesting the first text line at least 5 character long (i.e. a character in column position 5).
_11_BUF	When a column position 11 is reached and the character at position 11, 12 and 13 equal 'B', 'U', and 'F', a match is found.

CHARACTER-CLASS ('[') Meta Character

The meta character '[' signals to ed that the characters following up to the next meta character ']', forms a character class, that is, a text pattern that matches any single character from the bracketed list. For instance, [al] matches an 'a' or 'l', [a-z] matches any lower case letter (see the enclosed character set table for character sequencing). The ESCAPE meta character can also be used inside the character class if the character class contains a '^', '-', '@', or ']'. The NOT-CHARACTER-CLASS character ('^') is used to match any character not specified between the '[' and ']' and if it is not immediately after the '[', then it is treated as part of the character class.

PATTERN	MEANING
[AZ]	If the character to be matched is 'A' or 'Z', then a match is found.
[D-Fa-z]	Matched if the character is a 'D', 'E', or 'F' or any lower case letter.

[@--4]	Matched if the character is a '-', '.', '/', '0', '1', '2', '3' or '4'. See the ASCII character set table for the binary code assignment.
[^4-6L-Na-z]	Matched if the character is not a '4', '5', '6', 'L', 'M', 'N', or any lower case letter.
[p^y]	Matched if the character is a 'p', a '^', or a 'y'.

CLOSURE ('*') Meta Character

Any of the above pattern elements which attempt a single character match (literal character, ANY-CHARACTER ('?'), or character class) can be followed by the character '*' which then changes the single character match into a multiple character match. This process, called CLOSURE, is used to try to find the maximum number of consecutive occurences of that pattern (and zero occurences is a valid number of occurences). For example, a* matches zero or more a's; [a-z]* matches any string of zero or more lower case letters, etc.

PATTERN	MEANING
[0-9]*	Matches any integer of zero or more digits.
[a-z]*	Matches any lower case word
[A-Z][0-9A-Z]*	Matches any uppercase word if the word starts with an alphabetic character and is made up with alphanumeric characters (i.e. a FORTRAN type variable name).
(?*)	Matches any pattern between the parentheses.
??*	Matches an entire line of one or more characters.
b?*y	Matches any string which starts with a 'b' and ends with a 'y' such as 'boy', 'baby', 'baloney', 'bx y', etc.
aa*	Matches any string which contains one or more sequential a's such as 'aa', 'xaa',

'3aaaaaauu', etc.

A.3.2 Global Prefixes -- Include and Exclude

One of the most powerful features of ed is the GLOBAL prefix. It is used to position to selected text line(s) for the accompanying command (section A.1.4 COMMAND SUMMARY shows what commands can use the GLOBAL prefix) to process. The GLOBAL prefix may be preceeded by its own line designators and the prefix can be made to select lines that match or do not match the target pattern. Note that the pattern delimiters used by the GLOBAL prefix do not imply direction as they do for the text line pattern search (i.e. doing a G\pattern\ does not start searching backward; see 4.3.1 FORWARD\BACKWARD PATTERN SEARCHING for more information). The search is always forward and the delimiters are only used to define the target pattern for the GLOBAL prefix.

The GLOBAL prefix will make two full passes thru the text file. It will first check each text line and mark it if there is a match of the prefix target pattern. Then after all the text lines have been checked, ed will position itself to each of the marked text lines (making that text line the current text line) and execute the accompanying ed command.

EXAMPLE	MEANING
G/DATA/P	Locate all text lines with the word 'DATA' in it and output the text line to the terminal.

\SUBROUTINE X\,/END/X/%C/D

Starting from the current line number, search backwards until the 1st occurrence of 'SUBROUTINE X' is found and delete all lines that do not have a 'C' at the first character position until a 'END' is found. This statement is useful in extracting the comments from a FORTRAN subroutine.

G/%/M 0	To show the power of the GLOBAL prefix, this statement will get a line from

the text file and put it in line # 0 until the entire file is processed. The file will then be in backward order, the last line being first, the second-to-the-last line being second, etc. up to the first line being last.

A.3.3 Types of Commands

There are 6 different types of commands that are the mainstay of ed:

The Text File commands are used to describe and manipulate MP32 disk files.

The Text Line Environment commands are used to describe the environment in which ed works.

The Text Line Inspection command is used to display text line(s).

The Text Line Locating commands is used to help the user locate desired text lines.

The Text Line Manipulation commands are used to manipulate complete text line(s) of a text file.

The Individual Text Line command (SUBSTITUTE) is used to manipulate parts of individual text line(s).

A.3.4 Postfixes -- Global and Print

Postfixes are user specified options which are used to request subsequent sub-actions to be performed either during or after the command is processed. There are two postfixes which can be specified. The first is the letter 'G' (global) postfix which is explicitly used by the SUBSTITUTE command (see section A.5 SUBSTITUTE command for more information) for the substitution of all matches in a text line. The

second postfix is the letter 'P' which is used to request ed to print out the text line after the command is executed. If both the 'G' and 'P' are specified in the SUBSTITUTE command, the 'G' must come first (i.e. the postfix 'PG' is illegal).

EXAMPLE	MEANING
S/A/B/GP	Change all A's in the text line into B's and then print the changed line.
.k$p	Copy this line to the end of the file and print the copied line.

A.4 TEXT LINE COMMANDS

The following types of commands are used to describe and control the individual or parts of individual text line of a file.

The Text Line Environment commands are used to describe the environment in which ed works. The LENGTH command is used to designate or describe how much or which portion of the text line to display. The TAB command is used to display or set up column tabs which can be used when inputting text lines. The UPPER command requests that all lower case characters be treated as uppercase. The ZONE command is used to restrict the column positions that are used for the locating of desired text line(s). And the '=' command is used to request the display of the text line's line number.

The Text Line Inspection command is used to display text line(s).

The Text Line Locating commands help the user locate text lines in two ways. One is to use the carriage return key to the display of the next text line in the file, and the other is to request the first text line which matches a particular pattern.

The Text Line Manipulation commands are used to manipulate entire text line(s) of a file. The APPEND command is used to input text line(s) and place them after the current text line. The CHANGE command is used to replace existing text line(s). The DELETE command is used to remove specified text lines. The INSERT command is similar to the APPEND command

except that the inputted text line(s) are placed before the current text line. The KOPY command is used to copy existing text line(s) to any location in the text file. And the MOVE command is used to move text line(s) around in the text file.

The Individual Text Line command (SUBSTITUTE) is used to manipulate parts of individual text line(s). It can insert, append and change any desired part of the text line(s) with any replacement string (which includes the NULL string, which if used would delete that part of the line).

A.4.1 Text Line Environment Commands

The Text Line Environment commands are used to describe the environment in which ed works. Most of these commands involve specifying column positions. These column position numbers are used to denote character position in a text line, the 1st legal character position being column position 1, the 2nd character position being column position 2, etc.

A.4.1.1 Length Command

The LENGTH command is used to allow the user to display more or a different part of a text line other than the 1st 80 characters of a text line (text lines can be up to 136 characters in length). If the LENGTH command is entered with no column positions, then the starting and ending display column positions are shown. If one column position is specified and less than 137, then this is treated as the ending display column position and the starting display column position is set up as 1. If two column positions are entered, then the 1st number (which must not be greater than the 2nd column position number) becomes the starting display column position number and the 2nd column position number (which must be less than 137) becomes the ending display column position.

EXAMPLE	MEANING
L	If the display column positions are currently 1 and 80, then '1,80' is displayed.

L 40	Set up the starting display column position as 1 and the ending display column position as 40.
L 10,130	Set up the starting display column position as 10 and the ending display column position as 130.

A.4.1.2 Tab Command

The TAB command is used to display or set column tabs which can be used when inputting text lines and are very similar to those on a typewriter. Everytime ed sees a tab character in an input text line (see section A.4.4 TEXT LINE MANIPULATION COMMAND for more information on Input mode), it will position to the next column position. If there are no more column positions, then the tab character is treated as a normal character.

If the TAB command is entered with no parameters, then the current tab character and the tab column position(s) are displayed if tabbing is in effect (otherwise 'OFF' is displayed). If there is a non-numeric character included with the TAB command, then this character will be recognized as the tab character. If the TAB command is followed by a '0' (zero), then tabbing is disabled, the current tab character is reset to ';' and all tab positions cleared. If tab column positions are to be set, they must be a list of numerically ascending integers, separated by commas with the highest tab column position less than 137. Up to 10 tab column positions may be entered.

EXAMPLE	MEANING
T 0	Disable tabbing and reset the tab character and tab column positions.
T	Display the tab character and the tab column positions ('OFF' is displayed if tabbing is disabled).
T X	Set the tab character as 'X'.
T 10,20,41	Set TABS at 10, 20, 41.

The following is a session using the tabbing feature:

T	User request tabbing information.
OFF	Ed informs user that tabbing is disabled.
T \	User sets up tab character as '\'.
T 5,10,15	User sets up 3 tab column positions at 5, 10, 15.
T	User request tabbing information again.
\ 5,10,15	Ed displays current tab character and the 3 tab column positions.
A	User request ed to begin inputting text lines starting at the current text line position.
\A\B\C\D	First text line is entered.
1\12\130	Second text line is entered.
\0\20\10	Third text line is entered.
\ABCDEF\XY	Forth text line is entered.
.	Inform ed that no more text lines are to be inputted.

The inputted text lines would appear as follows:

```
          1         2
12345678901234567890

    A    B    C\D
1   12   130
    0    20   10
    ABCDEF    XY
```

A.4.1.3 Upper Command

The UPPER command is used to select either a 64 or 96 character set. When UPPER, carriage return is entered, either a 'ON' or 'OFF' is displayed (everytime an 'UPPER' command is entered, it will toggle between 64 and 96 character set). If 'ON' is displayed, then a 64 character set is in effect. If 'OFF' is displayed then the 96 character set is in effect.

A.4.1.4 Zone Command

The ZONE command is used to restrict the column positions that are used for the locating of desired text line(s). Normally, when a pattern match is attempted for a text line, the search is started at column position 1 and continues until the end of the text line. This restriction of the search to a particular section of a text line will speed up the execution of the match.

The ZONE command uses no line designators and if the ZONE command is entered with no parameters, then the search range is displayed. If only one number is included, then this number (which must be less than 137) is treated as the ending search column position and the starting search position is set up as 1. If two numbers are included, then the leftmost number must not be greater than the rightmost number (which must be less than 137) and are used to define both the starting and ending search column positions, respectively.

EXAMPLE	MEANING
Z	Display the current starting, ending search column positions.
Z 10	Set the starting search column position as 1 and the ending search column position as 10.
Z 10,20	Set the starting search column position as 10 and the ending search column position as 20.
Z 15,15	Restrict searches to column 15 only.

A.4.1.5 Display Current Line Number

Ed numbers the text lines of a text file. If the line number of a text line is desired, it can be displayed by entering '=', carriage return. It can have up to one line designator to position to a particular text lines and if none is given, then the current text line's line number is displayed.

EXAMPLE	MEANING
=	Display the line number of the current text line.
/SUBROUTINE/+1=	Locate the text line with the string 'SUBROUTINE' and display the line number of the next text line.

A.4.1.6 Repeat Last Ed Command ('-')

Ed has the ability to remember certain commands and repeat then whenever a '-', carriage return is entered. The commands shown in the HELP display and the Command Summary (Section A.1.4) that contain a dash ('-') in their syntax are repeatable. For example, this feature may be used to do a substitution, move to the next desired text line, and execute the same substitution by typing a dash, carriage return. The repeatable command is remembered until another repeatable command is entered.

A.4.2 Text Line Inspection Command -- Print

The Text Line Inspection or Print command is used to display text line(s) and can have up to 2 line designators (see section A.3.1 LINE DESIGNATORS for more information). If no line designator is given, then the current text line is displayed (if there is no current text line, then at least one line designator must be given). If one line designator is given, then the first line specified by the line designator is displayed. If two line designators are given, then the text line specified by the leftmost line designator to the text line specified by the rightmost line designator is displayed. The current text line is always the last text line displayed and the BREAK key can be used to terminate

further display of text line(s).

EXAMPLE	MEANING
P	Display the current text line.
\END\+1P	Search backward to the text line with the most previous occurence of 'END' in it and display the text line after it.
/SUB/+1,/END/-1P	Search forward to the next text line with the next occurence of 'SUB' in it and display the text lines after it, up to but not including the text line with 'END' in it.

A.4.3 Text Line Locating

Text line locating is the method used to search thru a text file to find a desired text line. There are two methods to locate and display text lines: the carriage return and the forward/backward pattern search. Using the carriage return method allows the user to display individual text lines sequentially. The use of the pattern search allows for the forward/backward, text line by text line pattern match, which completes when the first matching text line is found.

Sometimes there will be a time delay, a possible garbed display of a text line or the message 'No read from linked Task'. This is perfectly normal and is due to the heavy usage of the operating system or the disk. Eventually, ed should come back and act normally.

A.4.3.1 Carriage Return

When just a carriage return is entered , the next text line is displayed (i.e. a carriage return is equivalent to the command .+1p -- see section A.4.2 TEXT LINE INSPECTION COMMAND for more information). In using the carriage return to step thru a text file, the text lines will be displayed in ascending line number order and if the carriage return is encountered when at the last valid text line, an error is

issued. The current text line is the last text line displayed.

A.4.3.2 Forward/Backward Pattern Searching

The forward/backward pattern search involves defining a pattern which will uniquely describe the text line to be located (see 3.1.3 STRING PATTERNS for more information and examples). This pattern is saved in the pattern buffer and remains there until overwritten. If the pattern is enclosed with /'s, then the search is forward (numerically ascending text line numbers) and is backward if the pattern is enclosed with \'s. If a '//' or '\\' is entered with no intervening pattern, then the pattern in the pattern buffer is used.

The pattern search continues until either the text line is found or the search wraps back to the current text line. The ZONE command (see section A.4.1.4 ZONE command) can be used to define a smaller search area to assist in text line locating and displays the matching text line when found. So far, the discussion has been on locating the next pattern in line, but multiple pattern search can be entered together on a single line if the searches are separated by a ';'. For instance, the command /a/;/a/+1d would locate to the second occurance of the text line with an 'a' in it and delete the text line after it.

EXAMPLE	MEANING
/@/DEL/	Search foward and display the text line with '/DEL' in it.
\DA??\	Search backward and display the most recent text line with a four character string with 'DA' as the first 2 characters.
//	If the pattern buffer contains the pattern 'DJK', then serach forward and display the next text line with 'DJK' in it.

A.4.4 Text Line Manipulation Commands

The Text Line Manipulation commands are used to work on complete text line(s) of a file. These commands can have one or two line designators (see 3.1.0 to 3.1.3 LINE DESIGNATORS for more information) and the KOPY and MOVE commands use a destination line designator as well.

The APPEND command is used to input text line(s) and place them after the current text line. The CHANGE command is used to replace existing text line(s). The DELETE command is used to remove text line(s). The INSERT command is identical to the APPEND command except that the text line(s) are placed before the current text line. The KOPY command is used to copy existing text line(s) to any position in the text file. And the MOVE command is used to move text line(s) around in the working text file.

Ed has two modes of operation: Command mode and Input Text mode. Command mode is the processing of ed commands and Input Text mode is the entering of text line(s) into the working text file. When in input mode (which is entered with the APPEND, CHANGE, OR INSERT commands), text lines cannot be greater than 160 characters in length and be terminated by a carriage return. The TAB character is recognized only in input mode and input mode is terminated when a text line with a '.' (period) in column position 1, followed by a carriage return.

A.4.4.1 Append Command

The APPEND command is one of the three commands which is used to enter Text Input mode. It can have one line designator (see section A.3.1 LINE DESIGNATORS) to locate to a desired text line and all entered text lines will be placed after this text line (if there is no line designator, the text lines are placed after the current text line). Text lines that are inputted can be up to 160 character in length and text input mode is terminated when a '.' (period), carriage return is detected in column position one. At that point, input text mode is terminated and ed Command mode is entered. The last text line entered becomes the new current text line.

Assume that in the example, the working text file contains the following text lines:

```
SUB A
DATA 1              <This is the current text line>
```

```
DATA 2
END Y
SUB B
DATA A
DATA B
DATA C
END Z
```

Then after the following commands:

INPUT LINE	MEANING
A	Go into input text mode and begin appending text lines after text line 'DATA 1'.
TEXT 1	First text line is entered.
TEXT 2	Second text line is entered.
.	Inform ed that no more text lines are to be inputted.

The working text file would appear as follows:

```
SUB A
DATA 1
TEXT 1
TEXT 2                <This is the current text line>
DATA 2
END Y
SUB B
DATA A
DATA B
DATA C
END Z
```

A.4.4.2 Change Command

The CHANGE command is used to replace one or more existing text lines with new text line(s). The number of text lines to replace is specified by the line designators (see section A.3.1 LINE DESIGNATORS for more information) and text input mode is terminated when a text line with a '.' (period), carriage return in the first column position is detected by ed. If the number of replacement text lines is less than the

difference between the ending and starting line designators, the rest of the text lines will be deleted. If the number of the replacement text lines specified is greater than the line designator difference, the additional text lines will be inserted. The last text line entered becomes the new current text line and if there was no new line entered, then the next (or last) valid text line becomes the new current text line.

Assume that in the example, the working text file contains the following text lines:

```
SUB A
DATA 1              <This is the current text line>
DATA 2
END Y
SUB B
DATA A
DATA B
DATA C
END Z
```

Then after the following commands:

INPUT LINE	MEANING
C	Go into input mode and replace the current text line with the following text lines:
LINE A	First text line is entered.
LINE B	Second text line is entered.
.	Indicate no more inputting of text lines

The working text file would appear as follows:

```
SUB A
LINE A
LINE B              <This is the current text line>
DATA 2
END Y
SUB B
DATA A
DATA B
```

```
DATA C
END Z
```

Using the original text lines of the working text file:

INPUT LINE	MEANING
/DATA 1/,/DATA 2/C	Change the text lines starting with the next text line with a 'DATA 1' in it to the next text line with a 'DATA 2' in it with:
DATA Z	First text line is entered.
.	Finished entering text lines

The working text file would appear as follows:

```
SUB A
DATA Z              <This is the current text line>
END Y
SUB B
DATA A
DATA B
DATA C
END Z
```

A.4.4.3 Delete Command

The DELETE command is used to remove unwanted text line(s) from the working text file. The number of text lines to delete is specified by the line designators (see section A.3.1 LINE DESIGNATORS for more information) and unlike the text line locating pattern in section A.4.3, the search text line search will not be wrapped around. The next valid text line after the deleted text (or the last valid text line if the last text line was deleted) becomes the current text line.

If no line designator is included, then only the current text line is deleted. If one line designator is given, then the text line which the line designator specifies is

deleted. If there are two line designators, then the leftmost designator specifies the starting text line, the rightmost designator the ending text line to delete. If the Print postfix is given (see section A.3.4 POSTFIXES), then the new current text line is displayed.

Assume that in the following examples, the working text files contain the following text lines:

```
SUB A
DATA 1
DATA 2
END Y
SUB B
DATA A
DATA B              <This is the current text line>
DATA C
END Z
```

EXAMPLE	MEANING
D P	If the current text line is 'DATA B', then 'DATA B' is deleted and text line 'DATA C' becomes the current text line and is displayed.
/END/+2D	If the current text line is either 'SUB B', 'DATA A', 'DATA B', or 'DATA C' or 'END Z' and error will be issued. Otherwise text line 'DATA A' is deleted and text line 'DATA B' becomes the current text line.
\SUB\,/END/D P	If the current text line is either 'SUB A' or 'END Z', and error will be issued because 'SUB' or 'END' will not be found (wrap around is not in effect). If the current text line is either 'DATA 1', or 'DATA 2', then text lines 'SUB A', 'DATA 1', 'DATA 2', 'END Y' is deleted and text line 'SUB B' becomes the current text line and is displayed. If the current text line is either 'END Y' or 'SUB B', then whole text file is deleted and there is no current text line to display.

If the current text line is either 'DATA A', 'DATA B', or 'DA C', then 'SUB B', 'DATA A', 'DATA B', 'DATA C' is deleted and text line 'END Y' becomes the current text line and is displayed.

A.4.4.4 Insert Command

The INSERT command is identical to the APPEND command (see section A.4.4.1 APPEND for more information and example) except that text lines are inserted BEFORE the current text line. Using the APPEND example, an INSERT command instead of an APPEND command would make the working text file appear as follows:

```
SUB A
TEXT 1
TEXT 2              <This is the current text line>
DATA 1
DATA 2
END Y
SUB B
DATA A
DATA B
DATA C
END Z
```

A.4.4.5 Kopy Command

The KOPY command is used to copy existing text line(s) from one section of the working text file to another. It can have up to two line designators (see section A.3.1 LINE DESIGNATORS) to define the text line(s) to be copied and a destination line designator to specify where to copy to.

If no line designator is given, then the current text line is copied after the text line specified by the destination line designator. If one line designator is given, then the text line specified by the line designator is copied after the text line specified by the destination line designator. If two line designators are given, the section of text lines specified is copied after the text line specified by the destination line designator. If the Print postfix is given,

then all copied text lines are printed as copied and the current text line is the last text line copied.

Assume that in the example, the working text file contains the following text lines:

```
SUB A
DATA 1              <This is the current text line>
DATA 2
END Y
SUB B
DATA A
DATA B
DATA C
END Z
```

Then after the following commands:

INPUT LINE	MEANING
.K4	Copy the current text line after text line number 4 ('END Y').

The working text file would appear as follows:

```
SUB A
DATA 1
DATA 2
END Y
DATA 1              <This is the current text line>
SUB B
DATA A
DATA B
DATA C
END Z
```

Using the original text lines of the working text file:

INPUT LINE	MEANING
/END/K4	Assuming the current text line is 'DATA 1', then text line 'END Y' is copied after text line 'END Y'.

The working text file would appear as follows:

```
SUB A
DATA 1
DATA 2
END Y
END Y                <This is the current text line>
SUB B
DATA A
DATA B
DATA C
END Z
```

Using the original text lines of the working text file:

```
   INPUT LINE      MEANING

 /DATA A/,/DATA C/K/END Y/
```

Assuming the current text line is 'DATA 1', then text lines 'DATA A', 'DATA B', 'DATA C' is copied after text line 'END Y'.

The working text file would appear as follows:

```
SUB A
DATA 1
DATA 2
END Y
DATA A
DATA B
DATA C               <This is the current text line>
SUB B
DATA A
DATA B
DATA C
END Z
```

A.4.4.6 Move Command

The MOVE command is similar in action to the KOPY command (see Section A.4.4.5 KOPY command) for more information) except that sections of text line(s) are moved to the specified location in the working text file instead of being copied. If the Print postfix is used, only the last line

moved is printed.

Assume that in the example, the working text file contains the following text lines:

```
SUB A
DATA 1              <This is the current text line>
DATA 2
END Y
SUB B
DATA A
DATA B
DATA C
END Z
```

Then after the following commands:

INPUT LINE	MEANING
.M4	Move the current text line after text line number 4 ('END Y').

The working text file would appear as follows:

```
SUB A
DATA 2
END Y
DATA 1              <This is the current text line>
SUB B
DATA A
DATA B
DATA C
END Z
```

Using the original text lines of the working text file:

INPUT LINE	MEANING
/END/M$-2	Assuming the current text line is 'DATA 1', then text line 'END Y' is moved after text line 'DATA A'.

The working text file would appear as follows:

```
SUB A
DATA 1
DATA 2
SUB B
DATA A
END Y                <This is the current text line>
DATA B
DATA C
END Z
```

Using the original text lines of the working text file:

```
   INPUT LINE      MEANING

 /DATA A/,/DATA C/K/END Y/
```

Assuming the current text line is 'DATA 1', then text lines 'DATA A', 'DATA B', 'DATA C' is moved after text line 'END Y'.

The working text file would appear as follows:

```
SUB A
DATA 1
DATA 2
END Y
DATA A
DATA B
DATA C               <This is the current text line>
SUB B
END Z
```

A.5 Individual Text Line Command -- Substitute

The Individual Text Line command (SUBSTITUTE) is used to manipulate parts of individual text line(s). It can insert, append and change any desired part of the text line(s) with any replacement string by using two elements: A target pattern and a replacement string. Both the target patern and the replacement string are enclosed by a pair of delimiters

(a delimiter is any character not found in the target pattern or replacment string; used to define the the start and end of the pattern and/or string) and these have no implied direction, unlike the text line locating patterns (see section A.4.3.2 FORWARD/BACKWARD PATTERN SEARCHING).

The target pattern describes the portion of the text line that is to be substituted by the replacement string. This pattern is saved in the pattern buffer and remains until overwritten by another target or locating pattern (see section A.4.3.2 FORWARD/BACKWARD PATTERN SEARCHING and section A.3.2 GLOBAL PREFIXES -- INCLUDE AND EXCLUDE for more information). If there is no target pattern between the delimiters, then the pattern in the pattern buffer is used (if none, an error is issued). Everything described in section A.3.1.3 STRING PATTERNS is valid concerning target patterns, along with some special options when used with the SUBSTITUTE command. The '%' and '$' characters, when found in the target pattern are meta characters used to denote the beginning and end of the text line, respectively. The column position meta character is normally used to inform ed where to attempt the target pattern matching. If only the column position meta character is in the text pattern, then it can be used to locate to a particular column position in a text line for the insertion of the replacement string.

The replacement string is a string of characters that is to be substituted for the target pattern. If there is no replacement string between the delimiters, then the characters in the text line which match the target pattern will be deleted. The replacement string is composed mainly of characters, although there are two special meta-like characters which have special meaning. The '&' character, if found in the replacement string, represents the character string that matched the target pattern and that character string will become part of the replacement string. The '@N', '@n', or the line feed character is a NEWLINE character and instructs ed that the rest of the replacement string (or up to the next NEWLINE character) is to be treated as a new text line. This allows the insertion or the appending of a replacement string string text line either before or after the current text line.

The SUBSTITUTE command can have either a global postfix and/or up to two line designators to specify the text lines to be worked upon. If either are given, then only those text lines that have a target pattern match are substituted. If none are given, then only the current text line is tested for a match (if none are found, an error is issued). Note that only the first occurence of a target pattern match is substituted and if all occurences of a target pattern are to be substituted, then the global postfix (see section A.3.4

POSTFIXES -- Global and Print) must be given. The Print postfix (see section A.3.4 POSTFIXES -- Global and Print) can also be given to display the modified text line, but it must follow the global postfix option (i.e. the global postfix, if used, must preceed the print postfix).

EXAMPLE	MEANING
S/DATA A/INFO Y/	Assuming that the current text line contains a pattern 'DATA A', then the first occurence of that pattern is replaced by 'INFO Y'.
S/$/PQR/P	Append to the end of the current text line the string 'PQR' and then display it after the substitution.
S % ABC P	Insert at the beginning of the current text line the string 'ABC' and display it after the substitution.
S._10_.DATA.	Insert into column position 10 of the current text line the string 'DATA'.
S/ACTIVE/IN-&/P	Locate the string 'ACTIVE' and replace it with the prefix 'IN-' (making it 'IN-ACTIVE') in the current text line and display it after the change.

For the following examples, assume that the text file initially contains the following text lines:

```
1
SUB A
DATA 1              <This is the current text line>
DATA 2
END Y
1
SUB B
DATA A
DATA B
DATA C
END Z
```

After the command SUB A\,/END/S/%/C /the text file would appear as follows:

```
1
C SUB A
C DATA 1
C DATA 2
C END Y                <This is the current text line>
1
SUB B
DATA A
DATA B
DATA C
END Z
```

After the command X/%1$/S/%/ /the text file would appear as follows:

```
1
 SUB A
 DATA 1
 DATA 2
 END Y
1
 SUB B
 DATA A
 DATA B
 DATA C
 END Z                 <This is the current text line>
```

After the command /SUB/,/END/G/DATA/S//TEXT/P the text file would appear as follows:

```
1
SUB A
DATA 1
DATA 2
END Y
1
SUB B
TEXT A
TEXT B
TEXT C              <This is the current text line>
END Z
```

After the command \SUB\,/END/G/[23]*$/.-1,.S/%/*/P the text file would appear as follows:

```
1
SUB A
*DATA 1
*DATA 2          <This is the current text line>
END Y
1
SUB B
DATA A
DATA B
DATA C
END Z
```

A.6 FILE MANAGER ERRORS

Whenever there is an illegal file request, FILE MANAGER will return an error code to ed. Ed will display the error code as:

```
FILE MANAGER ERROR xx
```

The most common error code is 20, which indicates that the file does not exist. An interpretation of the remaining error codes may be found in the MPX/OS Reference Manual.

A.7 ED COMMAND SYNTAX DEFINITION

```
                    [ld]      Append
                              Batch      [filename[,[user][,[ed][,[akey][,dc]]]]]
                    [ld[,ld]]Change
[[ld,ld]<G|X>/text/][ld[,ld]]Delete      [P]
                              Edit       [filename[,[user][,[ed][,akey]]]]
                              Filename   [filename[,[user][,[ed][,akey]]]]
                              Help
                    [ld]      Insert
[[ld,ld]<G|X>/text/][ld[,ld]]Kopy        [ld][P]
                              Length     [l[,r]]
[[ld,ld]<G|X>/text/][ld[,ld]]Move        [ld][P]
                              Output     seq
[[ld,ld]<G|X>/text/][ld[,ld]]Print
                              Quit
                    [ld]      Read       [filename[,[user][,[ed][,akey]]]]
[[ld,ld]<G|X>/text/][ld[,ld]]Substitute/text/text/[G][P]
                              Tab        [<c|n[,n[,n[,n[,n[,n[,n[,n[,n]]]]]]]]]
                              Upper
[[ld,ld]<G|X>/text/][ld[,ld]]Write       [filename[,[user][,[ed][,akey]]]]
                              ZAp        [filename[,[user][,[ed][,akey]]]]
                              ZOne       [l[,r]]
                    [ld]      =
                    [ld]
```

ld(line designator) ::= .|$|number|/text/|\text\|ld<+|->ld

text ::= one or more ascii characters and/or meta-characters

meta-characters:

meta-escape	@
beginning-of-line	%
end-of-line	$
any-character	?
closure	*
column-position	_number_
character-class	[<cc|c-c|^c-c|^cc>]