PART 8

THE DOS/BATCH TEXT EDITOR

EDIT

PART 8

CHAPTER 1

INTRODUCTION TO EDIT A TEXT EDITOR

Controlled by user commands from the keyboard, the DOS/BATCH Text Editor (EDIT) creates and modifies ASCII source files. EDIT can merge or separate files through use of primary and secondary file commands. EDIT has I/O facilities which allow the user to: read or write any ASCII file to and from any device, list and verify portions of the text, and output form feed and null characters to the paper tape device. EDIT also has pointers for delimiting sections of text for modification. EDIT has the capability of modifying text by inserting, deleting, or exchanging portions of text. In conjunction with text modification, EDIT has macro facilities which allow the user to repeat a command operation as often as desired.

EDIT moves sequentially through the text by any number of lines or pages. A page is text delimited by form feed characters. If there are no form feeds, EDIT fills core and uses that as a page. Since EDIT is a sequential page editor, the user cannot page backward. To go back to a previous page, it is necessary to close the file and restart the editing session. Pages are useful for they benefit the user by sectioning text for ease of reference.

PART 8

CHAPTER 2

OPERATING PROCEDURES

2.1 CALLING AND USING EDIT

To call EDIT, load DOS/BATCH (if it is not running already) and log in. Then type:

```
$RUN EDIT
```

Upon receipt of this command, DOS/BATCH loads EDIT and begins execution. EDIT first requires an input/output specification for the editing session. After being loaded and started, EDIT responds with the # character indicating a request for a command string. The user responds with a command string of the form:

```
#dev:file1.ext[uic],dev:file2.ext[uic]<dev:file3.ext[uic]/B,dev:file4.ext[uic]
```

In the above specification, file1 is the primary output file and file3 is the primary input file, while file2 and file4 are the secondary output and input files respectively. Note the following:

1. dev: is the device name and is optional; the system disk is assumed if no device is specified. The assumed device changes, however, as soon as a new device is typed. See Part 3 (Monitor).

2. filen.ext is the filename and extension of the appropriate input or output file.

3. [uic] is the user code for the owner of the file, and need not be specified if the file in question is the user's own file.

4. /B is the no-backup switch. If /B is specified, a backup file is not created. Otherwise, the backup facility is used. If the /B switch is used, it must follow the primary input file specification since there is no backup facility for secondary files.

   The /B switch is useful when working with large input and output files having the same names. Since the backup facility is automatic, the specified device may get filled with extraneous backup files.

5. A minimum of one output file must be specified; a maximum of two input and two output files may be specified. If two input files are specified, the first one specified is the primary input file (i.e., file3.ext). This ASCII file can only be read; it is the user's primary editing file. The first output file specified is the primary output file (i.e., file1.ext) and it is into this file that corrected text is written.

6. If the primary input and output files have the same name and reside on the same device, EDIT creates the file EDITOR.TMP for output. When the primary input file is closed, it is renamed with .BAK as its extension. The output file EDITOR.TMP is then renamed to the desired name. This backup facility is automatic, but it can be suppressed with the /B switch. The user should never create a file with the name EDITOR.TMP as a fatal error results if EDIT needs the name and it is not accessible.

7. EDIT allows text to be read or written from secondary files. Secondary files are used as utility files to ease the process of editing. They are controlled by commands distinct from primary input and output file commands. All secondary input/output commands begin with the letter E (ER, EW, EH, EP), while all primary input/output commands are one-letter commands.

When EDIT receives a syntactically correct dataset specification, it performs internal initialization, then types an * indicating its readiness to receive the first command.

Whenever EDIT receives a syntactically incorrect command, or cannot execute a command properly, it returns an error message of the form:

Xnnn

> where X is either an S (System error) or W (Warning) and nnn is a numerical error code. (See Appendix K.)

If the error occurs after the first command in a command string, EDIT follows the error code with a second line. The second line is a copy of the command string being executed, with a ? following the character where EDIT found the error. Those commands preceding the question mark are executed and those commands following the question mark are not executed. EDIT then prints an * and awaits another command string.

In most cases, EDIT performs no further action on receipt of an illegal command. In effect, the command string is truncated at the error point.

2.2 CREATING A NEW FILE

There are two methods for creating a file. The first is the preferred method for there is less human intervention as the system writes and closes all files.

The first method is general and does not require a primary input device. It involves use of the Insert command to open the file and insert text. The line feed prepares EDIT for another command. The EXit command is used to close the file (see Section 8-3.3.1.7 for specific details). The general procedure for this method is:

```
#dev:filenam.ext
*I  <CR>
   :
 (text object)
   :
 <LF>
*EX
#
```

The second method is used when the keyboard is explicitly specified as the primary input device. Text is input via Read or Next commands. The Form Feed is pressed to terminate the text object and prepare EDIT for another command. The two options for closing the file are:

1. Use the Write command followed by an End File command. See Section 8-2.3.1.6.

2. Use the Write and EXit commands followed by CTRL/C. This elicits a . to which the user responds by typing EN then pressing the RETURN and LINE FEED keys.

The second method is illustrated with both options for closing the file.

```
#dev:filel.ext<KB:
*R   <CR>
    .
    .
    .
  (text object)
    .
    .
    .
  <FF>
*/W
*EF
*EX
#
```

or

```
#dev:filel.ext<KB:
*N  <CR>
    .
    .
    .
  (text object)
  <FF>
*/W
*EX
↑C
.EN
  <LF>
```

2.3 RESTARTING AN EDIT SESSION

To restart EDIT, use the Monitor REstart command after issuing a CTRL/C.

```
*↑C
.RE
*
```

Note that the above method of restarting enables a user to stop the progress of a command execution if desired.

2.4 FINISHING AN EDIT SESSION

To finish an editing session, use the EX command. This restarts EDIT for the next job unless the keyboard (KB:) is explicitly specified as a primary input device. In this case, the following sequence is used.

```
*EX                Type the EXit command,
↑C                 the CTRL/C combination,
EN    <CR>         the EN command, press RETURN,
___   <LF>         press LINE FEED.
```

NOTE

> Any other way of terminating an edit session, including system or power failure, results in the input or output files' not being properly closed. A file not properly closed cannot be used until correctly closed by the Unlock switch in PIP (see Part 12). Furthermore, if the backup facility was being used, the user may have to unlock and, then, rename or delete EDITOR.TMP (see Section 8-2.1).

2.5 ERROR RECOVERY

In the course of editing a page of the program, it may become necessary to correct mistakes in the commands themselves. There are two special commands which do this:

1. Pressing the RUBOUT key removes the preceding typed character, if it is on the current line.

2. The CTRL/U combination (holding down the CTRL key and typing U) removes all the characters in the current line.

If, in the course of a command string execution, a serious error is discovered, the restart capability can be used to terminate the command.

2.6 PROCEDURE WITH THE LOW-SPEED PUNCH

If the low-speed punch is one of the output devices, EDIT pauses before executing any command that writes to the punch. The punch must be turned on at this time, after which pressing the LINE FEED key on the keyboard initiates the output (the key does not echo on the tape). Following completion of the operation, EDIT pauses again to allow the user to turn the punch off. When the punch has been turned off, press the LINE FEED key and EDIT returns to command mode.

PART 8

CHAPTER 3

COMMANDS

3.1 MODE OF OPERATION

EDIT operates in two modes: command mode and text mode. The mode of operation determines the action performed on the characters in a command string. Whenever EDIT prints an asterisk (*) on the terminal, it is waiting to receive a command string, and is in command mode. While most commands operate in command mode, there are eight commands which require additional text, referred to as the text object, to operate upon. For example, the command to insert text must be followed by the text object to be inserted. The eight commands which require text objects are Insert, Get, wHole, Edit wHole, Position, Edit Position, Change and eXchange. (Capitalized letters represent the actual commands typed.) There are two ways to provide the required text object for a command. If the text object is small enough, EDIT can accept text in command mode if it is separated from the rest of the command string by delimiters. If the text object is long or contains carriage return or line feed characters, then the user must cause EDIT to enter text mode.

3.1.1 Command Mode

EDIT's processing begins in command mode. When the user types a command string, no action occurs until the string is completed by pressing the RETURN Key (symbolized as <CR>). What happens next depends on the last character in the command string. If the last character in the command string is not a text command, EDIT expects to find any necessary text objects embedded in the command string and stays in command mode as it executes the command string.

Text can be accepted in command mode if the text contains no carriage return or line feed characters, and is small enough to fit on a single typed line with the command string. When EDIT finds a text command which is not the last command in the command string, it looks at the next character in the string as a text delimiter. All characters between this and the next occurrence of the delimiter are considered the text object for the command. Thus, any ASCII character which does not appear in the text object is a valid delimiter.

3.1.2 Text Mode

If the text object is lengthy or contains carriage return or line feed characters, it can be accepted only if EDIT is in text mode. To enter text mode, type the text command of interest as the last character in the command string. Then press the RETURN key. When EDIT executes this command, it enters text mode and accepts text input. It continues to accept text until the user terminates text mode and re-enters command mode by pressing the LINE FEED key (symbolized as <LF>).

The line feed and carriage return are both counted as characters even though they do not print. The line feed is considered to be part of the text unless it is the first character entered on a line in text mode. If a user wishes to terminate text mode, he should press the LINE FEED key. Note that typing the RETURN key always causes the physical return of the printing head to the beginning of the line, and automatically generates a line feed, thereby advancing the carriage to a new line. In text mode, the RETURN key serves these mechanical functions allowing the user to continue typing at the beginning of a new line while simultaneously entering a carriage return and line feed character into the text.

The following example enters text mode and inserts three lines of text into the page buffer. Note that EDIT signifies its return to command mode by printing an *.

```
* I<CR>
BLOOP:      MOV       #17ØØØ,(R3)+       <CR>
            BIT       (R4)+,MASK         <CR>
            BNE       BLOOP              <CR>
<LF>
 *
```

3.2 COMMAND SYNTAX

3.2.1 The Character Location Pointer (Dot)

Almost all EDIT commands function with respect to a movable reference point, Dot. This character pointer is normally located between the character most recently operated upon and the next character. At any given time, it can be thought of as "where EDIT is" in the text. As will be seen shortly, there are commands which move Dot anywhere in the text, thereby redefining the "current location" and allowing greater facility in the use of the other commands.

3.2.2 Mark

In addition to Dot, a secondary character pointer known as Mark also exists in EDIT. This less agile pointer is used with great effect to Mark or "remember" a location by moving to Dot and conditionally remaining there while Dot moves on to some other place in the text. Thus, it is possible to think of Dot as "here" and Mark as "there". The position of Mark, which is referenced by means of the argument @, is discussed in the following commands.

3.2.3 Character-Oriented Command Properties

Many EDIT commands are character-oriented; that is, the argument to the command specifies the number of characters in the Page Buffer the command is to act on. The number of characters specified by the argument n is the same in the forward (n) and backward (-n) direction. Carriage Return and line feed characters embedded between text lines are counted in character-oriented commands, and are indistinguishable from other characters.

3.2.4 Line-Oriented Command Properties

EDIT recognizes a line as a unit by detecting a line terminator in the text. This means that ends-of-lines (line feed or form feed characters) are counted in line-oriented commands. This is important to know, particularly if Dot, which is a character location pointer, is not pointing at the first character of a line.

In such a case, the argument n will not affect the same number of lines (forward) as its negative (backward). For example, the argument -1 applies to the character string beginning with the first character following the second previous end-of-line character and ending at Dot. Argument +1 applies to the character string beginning at Dot and ending at the first end-of-line character. If Dot is located in the center of a line, the -1 would affect 1-1/2 lines back and the +1 would affect 1/2 lines forward.

The following example illustrates line-oriented command properties with ↑ illustrating the location of DOT.

```
TEXT

CMPB ICHAR,#Ø33
BEQ  $ALT
CMPB I CHAR,#175
      ↑
```

The command -1L yields the listing:

```
BEQ  $ALT
CMPB I
```

The command +1L yields the listing:

```
CHAR,#175
```

3.2.5 The Page Unit of Input

Input files to EDIT can be divided into smaller, more manageable segments called "pages". A page is delimited by a form feed (CTRL/FORM on keyboard) in the source text whenever a page division is desired. Although the unit of output is the line, the unit of input to EDIT is the page. In order to facilitate editing, the user may divide the input file into small segments by inserting form feeds in desired places before output occurs. Since more than one page of text can be in the buffer at the same time, it should be noted that the entire contents of the Page Buffer is available for editing.

3.2.6 Arguments

Some EDIT commands require an argument to specify the particular portion of text to be affected by the command or how many times to perform the command. In other commands this specification is implicit and arguments are not allowed.

The EDIT command arguments are described as follows:

1. n stands for any number from 1 to 32767 (decimal) and may, except where noted, be preceded by a + or -. If no sign precedes n, n is assumed to be positive. If n is omitted, the default value is 1.

   Where an argument is acceptable, its absence implies an argument of 1 (or -1 if a - is present). The role of n varies according to the command with which it is associated.

2. Ø refers to the beginning of the current line.

3. @ refers to a Marked (designated) character location (see Section 8-2.2.2).

4. / refers to the end of text in the Page Buffer.

The function of the arguments is explained in further detail in Section 8-2.3 Commands.

3.2.7 Command Strings

A command string consists of one or more commands typed on the same line. Spaces are not allowed between a command and its associated argument, although spaces may be inserted between commands in a command string. The command string, including embedded text objects, must be less than 72 characters and is terminated by typing the RETURN key.

NOTE

Caution must be exercised when using spaces in text commands. If a space separates a text command from its object, the space is considered the delimiter. Hence:

1. I OBJECT B/L is legal.

2. I #OBJECT#B /L will use #OBJECT#B as the text object.

3. I !OBJECT!B/L is illegal because there is no second delimiter.

The following are all legal command strings:

```
*B/L B G#OBJECT# <CR>
*B/L BG#OBJECT#I#TEXT# B/L <CR>
*BSKB3L <CR>
*-3J -4C#TEXT# ØA L <CR>
```

When the user types a command string to EDIT, it is not processed until he presses the RETURN key. Upon receipt of the carriage return character, EDIT starts at the beginning of the command string and serially processes each command. If an error is encountered in the middle of the string, those commands preceding the error are executed and those commands following the command in error are not executed.

3.3 EDITING COMMANDS

3.3.1 Input/Output Commands

The I/O commands allow files to be created, opened, listed or closed. Pages of the files can be read into memory for processing. Once editing is completed and the page is written to the output file, that page of text is unavailable for further editing until the file is closed and reopened. Note that the output process does not cause Dot to move.

3.3.1.1 Read and Edit Read

Two ways of getting text into storage so that it can be edited are by means of the Read (R) and Edit Read (ER) commands. There are no arguments for the R or ER Commands. The command R reads a page of text from the primary input file; the command ER reads a page from the secondary input file. The read text is appended to the contents (if any) of the page buffer. Text is input until one of the following conditions is met.

1. A form feed character is encountered.
2. The page buffer is 128 characters from being filled, or a line feed is encountered after the buffer has become 5ØØ characters from being full.
3. The end of data is detected on the input device.

An error message (W3Ø3) is printed if the Read (or Edit Read) exceeds the core available or if there is no input available. The command stops executing and no text is lost. A warning message W311 is printed when the end of data is detected on the input device.

The maximum number of characters which can be brought into core with an R command is approximately 2ØØØØ characters for a 16K system. Each additional 4K of core allows approximately 8ØØØ additional characters to be input.

3.3.1.2 Write and Edit Write

The Write command moves lines of text from the text buffer to the primary output file: the Edit Write command moves text to the secondary output file.

Format	Function	Length of Character String Operated On
nW nEW	Writes on primary output file Writes on secondary output file	The character string beginning at Dot and ending with nth end of line.
-nW -nEW	Writes on primary output Writes on secondary output	The character string beginning with the first character following the (n+1)th previous end of line and terminating at Dot.
ØW ØEW	Writes on primary output Writes on secondary output	The character string beginning with the first character of the current line and ending at Dot.
@W @EW	Writes on primary output Writes on secondary output	The character string between Dot and the Marked location.
/W /EW	Writes on primary output Writes on secondary output	The character string beginning at Dot and ending with the last character in the Page Buffer.

Examples:

5W	Write the next 5 lines of text, starting at Dot, to the current output file.
-2W	Write the previous 2 lines of text starting at Dot, to the current output file.
B/W	Write the entire text buffer to the current output file.

3.3.1.3 Next

The Next command writes the current page buffer (followed by trailer if paper tape is the output medium and a form feed is the last character in the buffer) onto the primary output file, clears the buffer, and reads the next page from the primary output file into the buffer. It performs the sequence n times as the format nN illustrates. Next accepts only positive arguments (n) and leaves the pointers Dot and Mark at the beginning of the page buffer. An error code (W311) is printed if fewer than n pages are available in the input file and the command executed for the number of pages available. It is used to space forward in page increments through the input file.

Example:

2N	Write the contents of the current text buffer to the output file. Read and write the next page of text. Then read another page into the page buffer.

3.3.1.4 List

The List command prints the specified number of lines on the terminal. The format of the command is:

nL	Beginning at Dot print n lines on the terminal.
-nL	Beginning n lines prior to Dot and ending at Dot prints n lines on the terminal.
ØL	Prints the character string beginning with the first character of the current line and ending at Dot.
@L	Prints the character string between Dot and Mark pointer locations.
/L	Prints the character string beginning at Dot and ending with the last character in the page buffer.

List accepts all legal line-oriented arguments and does not move the pointer Dot. See Section 8-3.2.4 and Chapter 8-5 for examples.

3.3.1.5 Verify

The Verify command prints the current text line on the terminal. The position of the pointer Dot within the line has no effect and the pointer does not move. The command format is:

```
V
```

and there are no legal arguments. The V command is equivalent to a 1L (List) command.

3.3.1.6 End File

The End File command closes the primary output file to any further output and renames it if a backup file is to be created. It also closes the primary input file to any further read operations. The secondary output and input files remain open for further editing. The form of the command is:

```
EF
```

and there are no legal arguments. EF is used when the output file is to be closed as it stands, with no further output desired. Note that this command is useful to create a truncated output file from a large input file.

3.3.1.7 EXit

The EXit command is used to terminate one edit session and begin another. Edit outputs the page buffer, copies the remainder of the primary input file into the primary output file, closes all files, and then is ready to accept another dataset command string. The format of the command is:

```
EX
```

and there are no legal arguments.

Always end the editing session with the EX command. EX guarantees that all files are closed correctly. If an EF command was executed during the editing session, EX does not perform any input/output operations - it merely closes files. The last command of every editing session should be the EX command.

NOTE

Following an EX command, the user should always wait for the # character response before entering CTRL/C KILL to ensure the integrity of his file structure.

3.3.1.8 Form Feed and Trailer

The Form Feed and Trailer commands are used when the primary output device is paper tape. The format of the Form Feed command is:

```
F
```

with no arguments. It writes a form feed character and four inches of null characters into the primary output file. The format of the Trailer command is:

```
nT
```

where n indicates the number of times to write four inches of Trailer (null characters) on the primary output device.

3.3.2 Commands to Move the Location Pointer

3.3.2.1 Beginning

The Beginning Command moves Dot to the beginning of the page buffer. The command format is:

```
          B
```

and there are no arguments.

Example:

Assume the buffer contains:

```
     MOVB  5(R1),@R2
     ADD   R1,(R2)+
     CLR   @R2
     MOVB  6(R1),@R2
          ↑
```

↑ indicates the current location of the Dot pointer.

The command:

```
          *B
```

moves the pointer to:

```
          MOVB 5(R1),@R2
         ↑
```

3.3.2.2 Jump

The Jump Command moves the pointer over the specified number of characters in the page buffer. The form of the command is:

nJ	Moves Dot forward n characters.
-nJ	Moves Dot backwards n characters.
ØJ	Moves Dot to the beginning of the current line.
@J	Moves Dot to the location pointer Mark.
/J	Moves Dot to the end of the page buffer.

Jump treats the carriage return, line feed, and form feed characters the same as any other character, counting one buffer position for each.

Examples:

*3J	Moves Dot ahead three characters.
*-4J	Moves Dot back four characters.

3.3.2.3 Advance

The Advance command moves the pointer the specified number of lines and leaves it at the beginning of the line. The form of the command is:

nA	Advances Dot forward past n ends-of-lines to the beginning of the succeeding line.
-nA	Moves Dot backward to the first character following the (n+1)th previous end-of-line.
ØA	Moves Dot to the beginning of the current line.
@A	Moves Dot to the location pointer Mark.
/A	Moves Dot to the end of the page buffer.

Notice that while n moves Dot n characters in the Jump command, its role becomes that of a line counter in the Advance command. However, because Ø, @, and / are absolute, their use with these commands overrides the line-character distinctions. Consequently, Jump and Advance perform identical functions if both have either Ø, @, or / for an argument.

3.3.2.4 Mark

The Mark command marks ("remembers") the current position of Dot for later reference in a command which uses the argument @. The format of the Mark command is:

```
M
```

and there are no arguments. Note that only one position at a time can be in a Marked state. Mark is also affected by the execution of the following commands: C (Change), D(Delete), H (wHole), I (Insert), K(Kill), N (Next), R(Read), X (eXchange), EH (Edit wHole), ER (Edit Read).

3.3.3 Search Commands

3.3.3.1 Get

The Get command starts at Dot and searches the current page buffer for the nth occurrence of the specified text string. If the search is successful, the Dot is left immediately following the nth occurrence of the text string. If the search fails, Dot is located at the end of the page buffer and an error message W3Ø7 is printed on the terminal before EDIT prints an *. The format of the command is:

```
nG (text)
```

where n is a positive integer. If no argument is present, n is assumed to be 1. If the text object is to follow the G command in the command string, it must be properly set off by delimiters. If G is the last character typed in the command string, EDIT enters text mode and accepts a search object of up to 72 characters.

Examples:

Assume the buffer contains:

```
↑MOV    PC,R1
 ADD    #DRIV-.,R1
 MOV    #VECT,R2
 CLR    @R2
 MOVB   5(R1),@R2
 ADD    R1,(R2)+
 CLR    @R2
 MOVB   6(R1),@R2
```

1. GADD positions the pointer at ADD↑#DRIV-.,R1

2. 3G$@R2$ positions the pointer at:

```
ADD R1,(R2)+
CLR @R2↑
```

3.3.3.2 wHole

The wHole command starts at Dot and searches the entire primary input file for the nth occurrence of the text string. If the nth occurrence of the text string is not found in the current page buffer, a Next command is automatically performed and the search continues on the new text in the buffer. The search proceeds until the search object is found or until the complete source text has been searched. In either case, Mark is located at the beginning of the page buffer. The format of the command is:

```
nH (text)
```

where the argument n, if specified, must be positive. If the search object (text) is found, Dot is located immediately following the nth occurrence of it.

If the search is unsuccessful, Dot is located at the end of the buffer and a W3Ø7 error message appears on the terminal. Upon completion of the command, EDIT is in command mode. Note that an H command specifying a nonexistent search object can be used instead of the EXIT command to copy all remaining text from the primary input file to the primary output file. The EXIT command returns control to the Monitor when execution is complete whereas the WHOLE command leaves control in EDIT when execution is complete.

3.3.3.3 Edit wHole

A wHole search can be performed through the secondary input file with the EH command. The EH search is identical to the WHOLE command except that the search is performed on the secondary input file and written into the primary output file. Note that an EH command specifying a nonexistent search object is a method of copying the entire secondary input file into the primary output file.

3.3.3.4 Position

The Position command searches the primary input file for the nth occurrence of the character string.

If the desired text string is not found in the current buffer, the buffer is cleared and a new page is read from the primary input file. The format of the command is:

```
          nP (text)
```

where the argument n, if specified, must be positive. When a P command is executed the current contents of the buffer are searched from Dot to the end of the buffer. If the search is unsuccessful, the buffer is cleared and a new page of text is read and the cycle continued.

If the search is successful, the pointer is positioned after the nth occurrence of the text. If the search fails, the pointer is left at the beginning of an empty text buffer.

The Position command is most useful as a means of placing the location pointer in the input file. For example, if the aim of the editing session is to create a new file out of the second half of the primary input file, a Position search saves time.

The difference between a wHole command and a Position command is that wHole writes the contents of the searched buffer to the output file while Position deletes the contents of the buffer after it is searched.

3.3.3.5 Edit Position

The Edit Position command is identical to the Position command except that the input file read and searched is the secondary input file. Note that any text in the buffer is written to the primary output file before the search begins. The format of the command is:

```
nEP (text)
```

where n is any positive integer (if specified).

3.3.4 Commands to Modify the Text

3.3.4.1 Insert

The Insert command inserts the specified text in the text buffer starting at the current pointer position. The location pointer (Dot) is positioned after the last character of the insert. The command format is:

```
I (text)
```

There are no arguments to the Insert command. In text mode, up to 8Ø characters per line are acceptable. Execution of the command occurs when the LINE FEED key (which does not insert a line feed character unless it is the first character typed in text mode) is pressed to terminate the text typed in text mode.

Dot is located in the position immediately following the last inserted text character. If Mark was anywhere after the text to be inserted, Dot becomes the new Marked location.

As with the Read command, an attempt to overflow the page buffer causes a W3Ø3 error message to be printed, followed by an * on the next line. This indicates that EDIT is ready to accept a new command; however, all or part of the last typed line may be lost. All previously typed lines are inserted. See Chapter 8-5 for examples.

3.3.4.2 Delete

The Delete command removes the specified number of characters from the text buffer. Characters are deleted starting at Dot; upon completion of the command, Dot is positioned at the first character following the deleted text. After execution of this command, Dot becomes the Marked location.

The format of the command is:

nD	Deletes the following n characters.
-nD	Deletes the previous n characters.
ØD	Removes the current line up to Dot.

@D	Removes the character string bounded by Dot and Mark.
/D	Removes the character string beginning at Dot and ending with the last character in the page buffer.

Examples:

Assume a buffer contains:

```
ADD  R1,(R2)+
CLR ↑@R2
```

1. ØD leaves the buffer as:

```
ADD R1,(R2)+
↑@R2
```

2. 3D leaves the original buffer as:

```
ADD  R1,(R2)+
CLR
```

3.3.4.3 Kill

The Kill command removes n lines from the page buffer. Lines are deleted starting at the location pointer. The pointer is then positioned at the beginning of the line following the deleted text. The command format is:

nK	Kills the character string beginning at Dot and ending at the nth end-of-line.
-nK	Kills the character string beginning from Mark (the first character following the (n+1)th previous end-of-line) and ending at Dot.
ØK	Removes the current line up to Dot.
@K	Removes the character string bounded by Dot and Mark.
/K	Removes the character string beginning at Dot and ending with the last character in the page buffer.

Examples:

Assume a buffer contains:

```
ADD  R1,(R2)+
CLR↑ @R2
MOVB 6(R1),@R2
```

1. /K alters the contents of the buffer to:

```
ADD R1,(R2)+
CLR
```

2.   -K   alters the contents of the original buffer to:

```
@R2
MOVB  6(R1),@R2
```

Kill and Delete commands perform the same function, except that Kill is line-oriented and Delete is character-oriented.

3.3.4.4 Change

The Change command replaces n characters, starting at Dot, with the specified character strings. The form of the command is:

nC (text)	Replaces the following n characters with the specified text.
-nC (text)	Replaces the previous n characters with the specified text.
ØC (text)	Replaces the current line up to Dot with the specified text.
@C (text)	Replaces the character string bounded by Dot and Mark.
/C (text)	Replaces the character string beginning at Dot and ending with the last character in the page buffer.

The Change command is identical to an Insert followed by a Delete which accepts all legal character-oriented arguments. If the Insert portion of this command is terminated because of an attempt to overflow the page buffer, data from the last line may be lost and text removal will not occur. Such buffer overflow might be avoided by separately executing a Delete followed by an Insert, rather than a Change which does an Insert followed by a Delete.

Examples:

Assume a buffer contains:

```
CMPB  ICHAR,#Ø33
BEQ   $ALT
CMPB  J↑CHAR,#175
```

1.   -1C /I/   Changes the last line of the buffer to:

```
CMPB  I↑CHAR,#175
```

3.3.4.5 Exchange

The Exchange command replaces n lines with the character string starting at the pointer. The form of the command is:

nX (text)	Replaces n lines with the specified text.
-nX (text)	Replaces the previous n lines with the specified text.
ØX (text)	Replaces the current line up to Dot with the specified text.
@X (text)	Replaces the character string bounded by Dot and Mark.
/X (text)	Replaces the character string beginning at Dot and ending with the last character in the page buffer.

The Exchange command is identical to an Insert followed by a Kill. As in the Change command, if the Insert portion of the command is terminated because of an attempt to overflow the page buffer, data from the last line may be lost and text removal does not occur. Such buffer overflow can be avoided by separately executing a Kill followed by an Insert rather than executing an Exchange.

3.3.5 Utility Commands

3.3.5.1 Save

The Save command copies the n lines beginning at Dot onto an external buffer called the Save buffer. The form of the command is:

```
nS
```

where n must be a positive integer because Save is a line-oriented command and operates only in the forward direction. Dot does not change nor is the saved data deleted. Any previous contents of the Save buffer are destroyed. If the Save command causes the Save buffer to exceed the core available, a W3Ø3 error message is printed and none of the text is saved.

3.3.5.2 Unsave

The Unsave command inserts the entire contents of the Save buffer at the location referenced by Dot. Dot then moves to the position following the last character Unsaved. The format of the command is:

```
U
```

The contents of the Save buffer are not destroyed by the Unsave command and may be Unsaved as many times as desired. If the action of unsaving would result in an overflow of the page buffer, a W31Ø error message is printed and the Unsave does not occur.

Note that Save and Unsave provide convenient tools for moving blocks of text or inserting the same block of text in several places.

3.3.5.3 Execute Macro

The Execute Macro command performs the same Edit command string n times. The format of the command is:

```
nEM
```

When the EM command is received, the contents of the save buffer up to the first carriage return character are interpreted as a command string and are executed n times. The macro is subject to the same rules as any typed command string, and in addition, a macro string may not contain another macro call. Thus, to execute a macro, it is necessary to insert the macro in the page buffer, save it, then execute it.

Example:

The following sequence of commands changes the first 15 occurrences of .CSECT in the buffer to .ASECT.

```
*BI
B G#.CSECT# -4J -C#A# <CR>
<LF>
*BSK15EM
```

3.3.5.4 Edit Open

The Edit Open command closes the secondary input file and reopens it at the beginning. The format of the command is:

```
EO
```

with no arguments.

Although EDIT is a one-pass editor, capable of making only one trip through the primary input file per job, with this command it is possible to make many passes through the secondary input file. EO has no effect on the text.

PART 8
CHAPTER 4
IMPLEMENTATION NOTES

4.1 MACRO USAGE

Use of the EM macro is most efficient if the text involved is small enough to fit within the macro itself. Large amounts of text can be inserted by a macro only if the macro is the first line of the Save buffer.

For example, suppose it is desirable to insert

```
          JSR  R5,RSAVE
          JSR  R5,RELOAD
```

after every occurrence of EMT 4Ø in the program. The text to be inserted is too long to enter in command mode, and there is no way to enter text mode from a macro, the Unsave command can be used.

To accomplish the above, the commands would look like:

```
          *BI
          G;EMT 4Ø; A U -3A K
               JSR    R5,RSAVE
               JSR    R5,RELOAD
          <LF>
          *B3S3K 5ØØØEM
```

The first command inserts the three lines needed to save into the Page Buffer. Note that the first of the three lines is the macro, while the last two are the code to be inserted. The macro itself contains the Get command to look for the EMT 4Ø, followed by an Advance command to advance Dot to the next line. The next command is the Unsave command, which will Unsave the three lines in the Save buffer. Since the macro is by definition the first line of the Save buffer, it is Unsaved along with the other two and has to be deleted. The -3A and Kill commands accomplish this. The second command string saves the macro and text object, deletes them from the Page Buffer, and executes the macro 5ØØØ times. Five thousand is an arbitrarily large number which assures reaching all the occurrences of EMT 4Ø in the buffer. When the search fails, the macro is halted and an error returned as Edit prepares to accept another user command string.

4.2 DELIMITER USAGE

When entering text in command mode, any ASCII character is acceptable as a delimiter; however, some characters are more appropriately used as a delimiter. The following are suggestions for choosing delimiters.

1. Use the same delimiter all the time (except when not possible because it appears in the text itself). Pick an uncommonly used character that is easy to type such as Q or ; and habitually using it will become second nature.

2. Avoid delimiters that are valid arguments or commands. Use of /, @ and valid commands as delimiters is an error-prone practice. Above all avoid using a space as a delimiter; forgetting that there is a space in the text object may yield erroneous results or an undesired command string (as illustrated below).

   For example:

   ```
   *G SAVE /DUMMY/ I$TEST$
   ```

   Although the user wanted to get "SAVE /DUMMY/", he will actually Get "SAVE", Delete to the end of the buffer, Unsave, Mark, Mark again, then receive an error message for the illegal Y command.

4.3 SUBSIDIARY I/O

Subsidiary I/O can serve several purposes. The following are suggestions for using subsidiary I/O.

1. If the secondary output file device is a line printer or video terminal, the user can quickly view his page buffer via the B/EW commands and check for mistakes.

2. Use of two output files is ideal for dividing a single input file into two smaller files.

3. If a user wants to move a very long section of text, or wants to save text for insertion later but may also want the Save buffer free, use of the HSR (wHole, Save, Read) and HSP (wHole, Save, Position) commands as subsidiary I/O is recommended. Punch the desired text, then read it in whenever desired.

4. Two input files are ideal for concatenation; do a wHole search for a nonexistent object through the primary input, then do the same through secondary input.

Remember, to close subsidiary files the EX command must be used.

4.4 CORE USAGE FOR SAVE AND UNSAVE

EDIT saves text during the Save command by setting aside a buffer large enough to accommodate the saved text. This decreases the total core available to the Page Buffer, resulting in the possibility that there might not be enough free core left to Unsave the text. EDIT guards against this situation by allowing the user to save only text that is short enough to guarantee at least enough room to insert it again. That is, the user is guaranteed enough room to Unsave at least once following every Save command that completes successfully. Of course, if the user Reads or Inserts a large amount of text between Save and Unsave, he might exceed the space available and not be able to Unsave. In this case, part of the buffer will have to be written into the output file to make room. A safe procedure is to do all Unsave's immediately following the corresponding Save commands.

PART 8

CHAPTER 5

EXAMPLES

I.

```
        #QUADRA.FTN

①       *I
                 REAL A,B,C
                 WRITE(1,20)
        20       FORMIT(' ENTER A,B,D')
                 READ(1,30) ↑A\↑A\A,B,C
        30       FORMAT(3F6.2)
                 ANS=((-B+SQRT(B**2-4*A*C))/2*A
                 ANSMIN=((-B-SORT(B**2-6*↑A\↑A\A*C))/2*A
                 IF(A.EQ.Ø) GOTO 999
                 ATEMP=(B**2-4*A*C)
                 IF (ATEMP.LT.Ø.) GOTO 999
                 WRITE(1,40)A,B,C,ANS,ANSMIN
        40       FORMAT(' THE ANSWER IS ',1Ø(1X,F8.2))
                 GOTO 199
        999      WRITE(1,5Ø)
        5Ø       FORMIT(' BAD DATA BYE!')
        199      STOP
                 END

②       *B
③       *2A
④       *V
        2Ø       FORMIT(' ENTER A,B,D')
⑤       *G/M/
⑥       *D
⑦       *I/A/
        *V
        2Ø       FORMAT(' ENTER A,B,D')
        *12A
        *V
        5Ø       FORMIT(' BAD DATA BYE!')
⑧       *G/M/
        *D
        *I/A/
        *V
        5Ø       FORMAT(' BAD DATA BYE!')
⑨       *13J
        *I/- /
⑩       *
        .V
        5Ø       FORMAT(' BAD DATA - BYE!')
⑪       *EX
```

II.

```
        #COMMEN.FTN

①       *I
        C        THIS PROGRAM SOLVES THE QUADRATIC EQUATION
        C        FOR SIMPLE INPUT.
        C        THE USER MUST ENTER A,B,C AND THE ANSWER IS RETURNED.

②       *EX
```

EDIT EXAMPLES

I. This is a very basic FORTRAN program for solving the quadratic formula. It is being developed at the terminal to illustrate an editing session. No responsibility is assumed for its accuracy or completeness.

1. The file QUADRA.FTN is created; the keyboard is the default device. The Insert command is used to supply text to the page buffer. The LINE FEED key is pressed to terminate input and force the editor response *. Notice that the LINE FEED is a non-printing character that advances the paper one line.

2. The Beginning command is issued to allow the user to begin the edit session at the top of the page buffer.

3. The 2A command advances the pointer 2 lines.

4. The V command prints the line to which the user is pointing for verification.

5. The Get command is used to find the first M in the string and place the character location pointer DOT after it. The / (slash) is used as a delimiter because it does not appear in the current character string.

6. The D command deletes the I in FORMIT.

7. The Insert command places the A where the I was in FORMIT, changing it to FORMAT as verified in the next line. The /(slash) is used as a delimiter because it does not appear in the current character string.

8. The following sequence of commands repeats the process of changing the I in FORMIT to an A for FORMAT, twelve lines down.

9. The 13J command moves the location pointer dot over 13 character positions. As Dot was originally located after the A in FORMAT from the last command, it is now positioned after the space and before the B in BYE.

10. A dash and a space are inserted before the B in BYE. The line is then verified.

11. The EX command is used to terminate the first editing session and to begin another.

II. The following program demonstrates that a file can be created to be merged with another file at a later time (see example IV).

1. The Insert command is used to input the following comments into the page buffer. The LINE FEED key is pressed to terminate text mode and elicit the editor's response.

2. EXit command is used to close the file and terminate the editing session.

III. #QUADRA.FTN<QUADRA.FTN

```
 1.   *H/20/V
              WRITE(1,20)
 2.   *AV
      20      FORMAT(' ENTER A,B,D')
 3.   *G/A,B,/DI/C/V
      20      FORMAT(' ENTER A,B,C')
 4.   *AV
              READ(1,30) A,B,C
 5.   *2AV
              ANS=((-B+SQRT(B**2-4*A*C))/2*A
 6.   *G./.I/(/3JI/)/V
              ANS=((-B+SQRT(B**2-4*A*C))/(2*A)
 7.   *AV
              ANSMIN=((-B-SQRT(B**2-6*A*C))/2*A
 8.   *G./.I/(/3JI/)/V
              ANSMIN=((-B-SQRT(B**2-6*A*C))/(2*A)
 9.   *AV
              IF(A.EQ.0) GOTO 999
10.   *3S
11.   *-2AV
              ANS=((-B+SQRT(B**2-4*A*C))/(2*A)
12.   *U
13.   *V
              ANS=((-B+SQRT(B**2-4*A*C))/(2*A)
14.   *-AV
              IF (ATEMP.LT.0.) GOTO 999
15.   *3AV
              IF(A.EQ.0) GOTO 999
16.   *M3AV
              WRITE(1,40)A,B,C,ANS,ANSMIN
17.   *@K
18.   *V
              WRITE(1,40)A,B,C,ANS,ANSMIN
19.   *-AV
              ANSMIN=((-B-SQRT(B**2-6*A*C))/(2*A)
```

III. The file QUADRA.FTN from example 1 is the input file; substantial edits are made, then the file is output as QUADRA.FTN. Since the /B switch is not used, the original QUADRA.FTN should be on the user area as QUADRA.BAK. This file can be used if the current file is accidentally destroyed or deleted.

1. The wHole command is used to fill the page buffer then search it for the first occurrence of 2Ø. The Verify command is used to print the line to verify where the 2Ø was found.

2. The next line is advanced to, then verified.

3. The Get command moves the location pointer to the comma after the B. Notice that more than one command can be juxtaposed. The D deletes the D in the character string then the Insert command adds the C. The entire line is then verified.

4. The location pointer is advanced to the next line and the entire line is verified.

5. Two more lines are advanced and the last one verified.

6. Notice the use of delimiters; the Get command is used to get the / (divide sign) which is delimited by . (periods). The Insert command in turn uses the / as a delimiter to insert the left parenthesis. The Jump command spaces over three characters to allow the Insert command to add the right parenthesis. The results of the entire command string are then verified.

7. After the user acknowledges the accuracy of the previous command string, the next line is advanced and verified.

8. Sequence #5 above is repeated.

9. The next line is advanced and verified.

10. The 3S command saves the three lines beginning with the line containing Dot (i.e., IF (A.EQ.Ø) GO TO 999). These three lines are stored in a temporary buffer which then aids in moving it to another location.

11. The -2AV moves the location pointer back two lines then verifies the line. This places the location pointer before the line being verified.

12. The Unsave command is issued to reinsert the three lines of text in the Save buffer back into the page buffer.

13. The line is verified. Dot is at the beginning of the three lines that were reinserted.

14. The previous line is verified.

15. The three lines are advanced, the next line is verified.

16. The location is Marked; three lines are advanced and the next line verified.

17. The @K removes the character strings bounded by Dot and Mark. The three lines that were inserted above the ANS= and ANSMIN= statements with the Save-Unsave commands are still located in their original position. The M@K eliminates the 3 lines from their original position.

18. The line containing Dot is verified.

19. The previous line is verified.

```
20.     *2AV
        40          FORMAT(' THE ANSWER IS ',10(1X,F8.2))
21.     *G/ER/I/S ARE/2DV
        40          FORMAT(' THE ANSWERS ARES ',10(1X,F8.2))
22.     *DV
        40          FORMAT(' THE ANSWERS ARE ',10(1X,F8.2))
23.     *G/,/2DI/5/V
        40          FORMAT(' THE ANSWERS ARE ',5(1X,F8.2))
24.     *BG/30/V
                    READ(1,30) A,B,C
25.     *AV
        30          FORMAT(3F6.2)
26.     *G/F/C-8-V
        30          F8RMAT(3F6.2)
27.     *-D
28.     *V
        30          FRMAT(3F6.2)
29.     *I-Ø-\-Ø\O-V
        30          FORMAT(3F6.2)
30.     *G/3F/C-8-V
        30          FORMAT(3F8.2)
31.     *B/L
                    REAL A,B,C
                    WRITE(1,20)
        20          FORMAT(' ENTER A,B,C')
                    READ(1,30) A,B,C
        30          FORMAT(3F8.2)
                    IF(A.EQ.Ø) GOTO 999
                    ATEMP=(B**2-4*A*C)
                    IF (ATEMP.LT.Ø.) GOTO 999
                    ANS=((-B+SQRT(B**2-4*A*C))/(2*A)
                    ANSMIN=((-B-SQRT(B**2-6*A*C))/(2*A)
                    WRITE(1,40)A,B,C,ANS,ANSMIN
        40          FORMAT(' THE ANSWERS ARE ',5(1X,F8.2))
                    GOTO 199
        999         WRITE(1,50)
        50          FORMAT(' BAD DATA - BYE!')
        199         STOP
                    END
32.     *G/ANSMIN/V
                     ANSMIN=((-B-SQRT(B**2-6*A*C))/(2*A)
33.     *G/6/-DI/4/V
                     ANSMIN=((-B-SQRT(B**2-4*A*C))/(2*A)
34.     *EX
```

IV.

```
        #QUADRA.FTN<COMMEN.FTN,QUADRA.FTN

1.      *R
2.      */L
        C            THIS PROGRAM SOLVES THE QUADRATIC EQUATION
        C            FOR SIMPLE INPUT.
        C            THE USER MUST ENTER A,B,C AND THE ANSWER IS RETURNED.
3.      *3AV
4.      *ER
```

20. Two lines are advanced and the next line verified.

21. Get finds ER, then Insert adds S ARE, then the next two characters are deleted and the entire line is verified.

22. The next letter following the Dot (i.e. the S in ARES) is deleted and the entire line verified again.

23. Get finds the comma, then deletes the next two characters to insert 5. The line is verified.

24. The Dot is moved to the Beginning of the page buffer and a search for the first occurrence of 3Ø is made, the line where it occurs is verified.

25. The line is advanced one and verified.

26. The search command to get the F and change it to an 8 are realized as causing rather correcting an error when the line is verified.

27. The 8 is deleted.

28. The line is verified.

29. The O is inserted back in FORMAT and the line verified. The command looks odd because the RUBOUT key was pressed to rub out the dash and zero and replace it with the letter "O" and a dash. The - (dash) is used as the delimiter. The RUBOUT key was pressed twice, consequently, the / to indicate a RUBOUT and the 2 preceding characters were echoed.

30. The Get command is used to position Dot after 3F then the Change command is used to change the 6 to an 8.

31. The Dot is placed at the Beginning of the page buffer then the entire buffer is listed.

32. Dot is still located at the beginning of the buffer; consequently, the search command Get can be used to find the first line containing ANSMIN. Line is then verified.

33. The 6 is found then deleted and the 4 is inserted in its place. The entire line is verified.

34. The EXit command is used to terminate the editing session, close all files, and get ready for another session.

IV. This example illustrates the merging of multiple files into a single output file.

1. The primary input file COMMEN.FTN is read into the input buffer.

2. The entire input buffer is listed.

3. The three lines are advanced and the next verified. Since there is no next line nothing is printed.

4. The secondary input file is read into the page buffer starting at Dot. This has the effect of merging the two files into one file.

```
(5.)    *B/L
        C          THIS PROGRAM SOLVES THE QUADRATIC EQUATION
        C          FOR SIMPLE INPUT.
        C          THE USER MUST ENTER A,B,C AND THE ANSWER IS RETURNED.
                   REAL A,B,C
                   WRITE(1,2Ø)
        2Ø         FORMAT(' ENTER A,B,C')
                   READ(1,3Ø) A,B,C
        3Ø         FORMAT(3F8.2)
                   IF(A.EQ.Ø) GOTO 999
                   ATEMP=(B**2-4*A*C)
                   IF (ATEMP.LT.Ø.) GOTO 999
                   ANS=((-B+SQRT(B**2-4*A*C))/(2*A)
                   ANSMIN=((-B-SQRT(B**2-4*A*C))/(2*A)
                   WRITE(1,4Ø)A,B,C,ANS,ANSMIN
        4Ø         FORMAT(' THE ANSWERS ARE ',5(1X,F8.2))
                   GOTO 199
        999        WRITE(1,5Ø)
        5Ø         FORMAT(' BAD DATA - BYE!')
        199        STOP
                   END
(6.)    *EX

        #
```

5. Dot is moved to the beginning of the page buffer and the entire buffer is then listed.

6. The file is closed and the editing session terminated.

PART 8

CHAPTER 6

COMMAND SUMMARY

In the following table, # represents any legal text delimiter. <CR> represents a return character, <LF> represents a line feed character, and n represents any number between 1 and 32767 (decimal).

Command	Format	Result
Advance	nA	Advance Dot n lines. Leave Dot at beginning of line.
Beginning	B	Move Dot to the beginning of the Page Buffer.
Change	nC#XXXXX# <LF> or nC<CR> XXXXX<CR> <LF>	Change n characters to XXXXX. Equivalent to Insert followed by n Delete.
Delete	nD	Delete n characters from text.
Edit Open	EO	Move to the beginning of the secondary input file. Must follow a W311 error message before an ER can be executed.
Edit Position	nEP#XXXXX# or nEP<CR> XXXX <LF>	Perform a Position search using secondary input rather than primary input file.
Edit Read	ER	Read from secondary input file until form feed encountered.
Edit wHole	nEH#XXXXX# or nEH<CR> XXXX <LF>	Perform a wHole search for the nth occurrence of XXXXX, using the secondary input and primary output files.
Edit Write	nEW	Write n lines into secondary output file.
End File	EF	Close the primary output file to any further output and close the primary input file.
eXchange	nX#XXXXX# or nX<CR> XXXXX<CR> <LF>	eXchange n lines for XXXXX. Equivalent to Insert followed by n Kill.
Execute Macro	nEM	Execute the first line of the Save Buffer as a command string n times.

Command	Format	Result
Exit	EX	Perform consecutive Next commands until EOM or EOF reached. Close all files, and return to # mode.
Form feed	F	Write form feed into primary output file.
Get	nG#XXXXX# or nG<CR> XXXX<CR> <LF>	Search for the nth occurrence of XXXXX. Return with Dot following XXXXX.
Insert	I#XXXXX# or I<CR> XXXXX<CR> <LF>	Insert the text XXXXX at Dot. Move Dot to follow XXXXX.
Jump	nJ	Move Dot over n characters.
Kill	nK	Kill n lines of text.
List	nL	List n lines on teleprinter.
Mark	M	Mark the current location of Dot.
Next	nN	Write the contents of the Page Buffer onto the primary output file, kill the buffer, and read a page of text from the primary input file. Repeat n times. Equivalent to B/W /D R.
Position	nP#XXXXX# or nP<CR> XXXX <LF>	Perform a Next command, then search for the nth occurrence of XXXXX. If found, return with Dot following XXXX. If not found, clear the buffer, read another page, and continue search.
Read	R	Read from primary input file until form feed encountered.
Save	nS	Save the next n lines in the Save Buffer.
Trailer	nT	Write 4∅ null characters as trailer on the primary output device if device is paper tape.
Unsave	U	Copy the contents of the Save Buffer into Page Buffer at Dot.
Verify	V	Verify the present line via teleprinter.
wHole	nH#XXXXX# or nH<CR> XXXXX<CR> <LF>	Search for the nth occurrence of XXXXX. If found, return with Dot following XXXXX. If not found, execute an N command and continue search.
Write	nW	Write n lines into primary output file.