SOFTWARE REFERENCE MANUAL

HDOS SYSTEM

Chapter 3

HEATH TEXT EDITOR EDIT

HEATH COMPANY
BENTON HARBOR, MICHIGAN 49022

595-2337
Printed in the United States of America

TABLE OF CONTENTS

INTRODUCTION

The Heath Text Editor (EDIT) converts your system into a very sophisticated typewriter. This typewriter is not only capable of generating text, but also has powerful editing capabilities. With these capabilities, even if you are a poor typist, you can create error-free text, organized as you desire.

EDIT is a very powerful utility program with many uses. You can use it to enter and edit assembly language programs and BASIC programs, as well as to create and edit reports, letters, and manuscripts. Note that EDIT supports the ASCII TAB character to conserve space in the text files.

You can do your editing by command, referencing the desired line, and text is stored in a section of memory called the **buffer**. When the buffer is full, you can transfer the text to a disk file. Additional text can be read in from previously created files, or you can place it in the buffer from the terminal keyboard. EDIT's file handling capabilities allow it to edit large files on a piecemeal basis.

All of the RAM in your system that is not used by HDOS or by the EDIT program is available in the buffer. A computer containing 16K of RAM has approximately 4K bytes for a buffer, which provides sufficient room for a well-documented, 300-line program. You can conveniently edit larger files by using the file handling capabilities of EDIT. Systems containing more RAM will have proportionally larger buffers.

EDIT has many unique features that are discussed in detail on the following pages. Some of these features are:

- 15 commands for text editing versatility.
- Terminal control of output and input operations.
- Command completion and command error analysis.

Character Set

EDIT supports the entire 96 character ASCII character set, including lower case characters, Form-Feed, and TAB. You may enter text and commands in lower case. With the exception of TAB (CTRL-I) and Form-Feed (CTRL-L), you may not use control characters in the text.

EDIT is called by HDOS as follows

```
>EDIT (CR)
EDIT ISSUE #103.00.00
—
```

EDITOR MODES OF OPERATION

Two modes of operation called the "Command Mode" and the "Text Mode" are available in EDIT. These two modes distinguish between editing commands and text being entered into the buffer.

The Command Mode

The command mode can be subdivided into three areas: input commands, output commands, and editing commands. You execute all commands by typing the appropriate command on the terminal, and follow this with a carriage return; this will be indicated throughout this reference Manual by the symbol (CR). In actual use, no symbol is printed when a carriage return is typed on the terminal, as the carriage or cursor simply moves to the first column of the next line. In the command mode, the prompt character - - (a double dash) appears in the first two columns.

The Text Mode

In this mode you can add text to the buffer from the terminal keyboard, the normal source for most text. Once a source file has been created, it can be stored on a file. Later, you can use this file as a source of text to be read into the buffer.

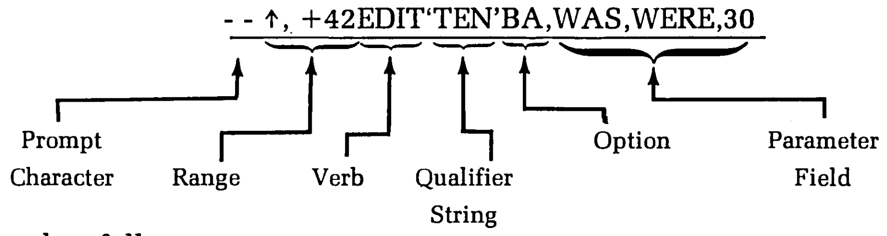
Type the CTRL-C key when you wish to return from the text mode to the command mode. This performs two functions. First, it discards the line of text which it was on when the key was struck. Second, it returns the Text Editor to the command mode. To preserve the last line of text, type a carriage return to generate a new line before striking the CTRL-C key.

THE COMMAND STRUCTURE

The basic commands for EDIT have the following form:

```
[<range>] [<verb>] [<qualifier string>] [<option>] [<parameters>]
```

The **range** indicates what lines in the buffer the command affects; the **verb** is the basic command. The **qualifier string** limits the command to those lines containing a given string, and the **option** permits you to view the line before or after (or both). The **parameters** (parameter field) contains specific instructions for some commands. For example, the command



is read as follows:

The lines starting with the first line (↑) and ending with the 43rd line (+42) are to be edited (EDIT). The EDIT command replaces one string with another. The affected lines are limited to those containing the string "TEN" (the optional **qualifier string**). The **option** (BA) indicates that you wish to view the lines before and after editing. And in this particular case, the **parameter field** specifies the old and new strings and the count. The word WAS is to be replaced by the word WERE, a maximum of 30 times. NOTE: EDIT string used in the **range expression** or **qualifier strings** must be enclosed in a single quote (') (apostrophe).

Range Expressions

The range expressions define the buffer lines on which the command is to operate. They may define a single line, or two expressions may be used to define the range over which the command works.

A range expression may consist of:

A line expression.
A multiple-line expression.
A null.
A blank.
An equal sign.

These different range expressions are explained below.

NOTE: Text must be in the buffer before a range expression will be accepted. If the text is not in the buffer, a bell code will sound as you try to complete the range expression. To use the following examples, use the INSERT command (see Page 3-12).

SINGLE-LINE EXPRESSIONS

Either one of the following expressions may be used to define a single line:

THIS SYMBOL	DEFINES
↑	The first line.*
$	The last line.

EDIT is always pointing to some line of text. Once you have explicitly pointed to a line by referencing the first line (↑) or the last line ($), you may make future line references by referencing to the current line pointer. NOTE: Once a DELETE has been executed, the current line pointer is reset and you must explicity reference a line to re-establish the line pointer.

A. + count. A plus (+) followed by a decimal number (n) refers to the nth line past the current line pointer.

B. – count. A minus – (n) refers to the nth line preceding the current line pointer.

C. + 'string'. The + 'string' refers to the first line in the text buffer which contains the designated 'string' after the current line pointer.

D. – 'string'. The – 'string' refers to the first line in the buffer, preceding the current line pointer, containing the indicated 'string'.

* NOTE: The uparrow character (↑) is produced on a H9 by holding down the shift key and typing N. On most keyboards, it is produced by holding down the shift key and typing 6.

For example, assume the buffer contains:

```
--INSERT (CR)
THIS IS THE FIRST LINE (CR)
MARY HAD A LITTLE LAMB (CR)
THIS IS THE THIRD LINE (CR)
ITS FLEECE WAS WHITE AS SNOW (CR)
THIS IS THE FIFTH LINE (CR)
AND EVERYWHERE THAT MARY WENT (CR)
THIS IS THE SEVENTH LINE (CR)
THE LAMB WAS SURE TO GO (CR)
THIS IS THE LAST LINE (CR)
^C                                                    (user typed CTRL-C)
```

EDIT responds as follows to the range commands.

```
--↑PRINT (CR)
THIS IS THE FIRST LINE
--$PRINT (CR)
THIS IS THE LAST LINE
--↑+2PRINT (CR)
THIS IS THE THIRD LINE
--↑+'FLEECE'PRINT (CR)
ITS FLEECE WAS WHITE AS SNOW
--$-1PRINT (CR)
THE LAMB WAS SURE TO GO
--$-'EVERYWHERE'PRINT (CR)
AND EVERYWHERE THAT MARY WENT
--
```

MULTIPLE-LINE EXPRESSION

Use a multiple-line expression when you want to define a group of lines to be operated on by the command. Use the comma as a delimiter to separate that start line from the stop line. The symbols ↑, $, + count, − count, + 'string', and − 'string' have the same meaning as they do with a single-line command. NOTE: A wide range of combinations may be used to identify the first and last lines of a multiple-line expression.

For example, you could print the contents of the previous buffer using these commands:

```
--↑,↑+3PRINT ⓒⓡ
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW

--↑+2,+3PRINT ⓒⓡ
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT

--↑+'FLEECE',+1PRINT ⓒⓡ
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE

--$-'THAT',+'SURE'PRINT ⓒⓡ
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO

--$-'THAT',+2PRINT ⓒⓡ
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
--
```

THE BLANK

When the verb is preceded by a single blank (space), the range is the entire buffer. For example, printing the entire buffer is accomplished by:

```
--△PRINT ⓒⓡ
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
```

Note the blank (space) between the prompt character (--) and the word PRINT. The blank is created by typing the space bar on the console terminal.

THE NULL

The NULL expression (the absence of any character) sets a single-line range at the first line of the previous command.

For example:

```
--↑+2,+'FIFTH'PRINT (CR)
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
--PRINT (CR)
THIS IS THE THIRD LINE
--
```

Note that there is no blank between the prompt (--) and the word PRINT.

```
--↑+2,+1PRINT (CR)
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
--,+2PRINT (CR)
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
--
```

NOTE: In this example, the NULL starts the range at the first line of the previous range, and the, +2 directs EDIT to print two additional lines.

THE EQUAL (=)

The = expression sets the range to the range of the previous command. For example:

```
—$'MARY',+'GO'PRINT (CR)
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
—=PRINT (CR)
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
—
```

Note that the command =PRINT causes the same lines to be printed in the first and second halves of the example.

The Verb

The verb specifies the action to be taken by the Editor. For example, the command PRINT or the command EDIT are verbs within the Text Editor's vocabulary.

All verbs are command completed. As soon as the Text Editor receives enough characters to know that only one command is possible, it prints the balance of the command without any additional keys being struck.

The verb will be refused if it is not valid for the current EDIT condition. For example, an attempt to PRINT an empty buffer is meaningless and the PRINT verb will be rejected.

For example, when you type the P key, the Text Editor knows that no other command begins with P. Therefore, the P is immediately followed by RINT. However, if you type the N, it does not know if the command NEWIN, NEWOUT, or NEXT is to be used. So the Editor prints NE and waits for a W or X. If it receives a W, it then waits for an I or an O. It completes these by filling in the N or UT. If it receives an X following the E, it then prints the T. A complete list of all of the command verbs and their specific actions and limitations follows in "The Commands" (Page 3-12).

The Qualifier String

The qualifier string is a further restriction upon the range expression. It is completely optional. If it is not indicated, it is not used.

The range expression may indicate work over a certain number of sequential lines, starting with a given line. The qualifier string further limits these lines to those within the range containing the string specified in the qualifier field. This string is enclosed in single quotes (') and contains all normal ASCII characters with the exception of the single quote (').

For example, to print the entire buffer (of the previous example), but only those lines with the string 'line':

```
—△PRINT'LINE' Ⓒⓡ
THIS IS THE FIRST LINE
THIS IS THE THIRD LINE
THIS IS THE FIFTH LINE
THIS IS THE SEVENTH LINE
THIS IS THE LAST LINE
```

The Option Field

The option field contains characters which let you view the line to be worked on, and/or the line after it has been worked on.

As its name implies, it is completely optional. You may insert any one of the following three option forms, or none at all.

1. B . . . . The BEFORE option displays the line **before** the command is executed.

2. A . . . . The AFTER option displays the line **after** the command execution.

3. BA . . . . This is a combination of the previous two commands. The line is displayed **before** and **after** command execution.

For example, presume that the buffer erroneously contained

```
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS RED AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
```

It may be edited to read correctly by

```
--↑+'RED'EDITBA,RED,WHITE,1 ⓒⓡ
ITS FLEECE WAS RED AS SNOW
ITS FLEECE WAS WHITE AS SNOW
--
```

Note that after the command EDIT, the options B & A are used to check the work.

NOTE: There are certain times when the command renders the option meaningless; DELETE BA, for example. The AFTER portion of the command is meaningless, as the line (or lines) cannot be displayed after deletion.

THE PARAMETER FIELD

This is a special field used with the EDIT, NEWIN, and NEWOUT commands. These commands require additional operating information, which is placed in the parameter field. The nature of this information depends upon the command used. The exact format is discussed under those commands.

THE COMMANDS

The following paragraphs give a complete description of each of the command verbs. Examples of many commands are also given to demonstrate some of the combinations of range expressions, qualifier strings, options, and parameter fields (if required) that may be used with this command. NOTE: All possible combinations of range expressions, qualifier strings, options, and parameter fields are not given. You must review the section for each of these expressions to determine all possible command structures.

INSERT

The INSERT command places EDIT in the text mode and is used to add text to the buffer from the console keyboard. This command adds text to the buffer on the next line following the first line of the range expression. The second line of the range expression is meaningless. Also, if the range expression is given as a line minus a count, the appending point is still the next line following the indicated line. The option, qualifier string, and parameter fields are ignored for the INSERT command. The range command "blank INSERT" is a special case that is used to insert a line before the first line in the buffer.

For example:

```
--△PRINT (CR)
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--


--$INSERT (CR)
THIS IS AN ADDITIONAL LAST LINE (CR)
<CTRL-C>
--


--↑+'FIFTH'INSERT (CR)
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE (CR)
<CTRL-C>
--


--△PRINT (CR)
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
THIS IS AND ADDITIONAL LAST LINE
--
```

NOTE: Use the CTRL-C key to return to the command mode. This will cause the current line to be erased. If you strike the CTRL-C key after inserting a partial line, that partial line will be lost. Therefore, to preserve the last line of inserted text, type a carriage return (this will create a new blank line in the text buffer) prior to typing the CTRL-C key.

REPLACE

This command causes the eligible line(s) in the command range to be replaced. Once the executing carriage return is typed, the terminal cursor moves to the start of the first line to be replaced. Type the replacement lines one at a time. Tabs, backspaces, and rub-outs may be used as part of the replacement text. However, typing a carriage return signifies replacement of another line within the command range.

After the lines indicated by the range expression have been replaced, EDIT reverts to the command mode. Typing CTRL-C returns the editor to the command mode, erasing the current line.

The qualifier strings and option may be used. There is no parameter field for the REPLACE command.

For example:

```
--△PRINT (CR)
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--
```

```
--↑+5REPLACEB (CR)
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
THIS LINE IS REPLACED (CR)
--
```

```
--↑+6,+'LAST'REPLACE'LINE' (CR)
THIS REPLACES THE OLD SEVENTH LINE (CR)
THIS REPLACES THE OLD LAST LINE (CR)
--
```

```
--△PRINT (CR)
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS LINE IS REPLACED
AND EVERYWHERE THAT MARY WENT
THIS REPLACES THE OLD SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS REPLACES THE OLD LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--
```

Note: Only lines containing the string 'line' are replaced in the second example, as the string 'line' is used as a qualifier.

DELETE

The DELETE command causes the eligible line(s) in the command range to be deleted. You may use the option B; however, the option A is meaningless. You may also use the qualifier string. There is no parameter field accompanying the DELETE command. You may not delete the entire buffer. To do this, use the BLITZ command.

For example:

```
--△PRINT (CR)
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS LINE IS REPLACED
AND EVERYWHERE THAT MARY WENT
THIS REPLACES THE OLD SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS REPLACES THE OLD LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--


--$DELETE (CR)
--$PRINT (CR)
THIS REPLACES THE OLD LAST LINE
--
```

```
—↑,$-1DELETEB (CR)
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS LINE IS REPLACED
AND EVERYWHERE THAT MARY WENT
THIS REPLACES THE OLD SEVENTH LINE
THE LAMB WAS SURE TO GO
—PRINT (CR)
THIS REPLACES THE OLD LAST LINE
—
```

EDIT

Use the EDIT command to replace one string with another. The string to be replaced and the new string are given in the parameter field of the EDIT command. The parameter field of the EDIT command takes the form:

```
<delimiter> old string <delimiter> new string <delimiter> <count>
```

The **delimiter** is an arbitrary delimiting character. (It may not be a carriage return). The **count** is a decimal count showing the number of replacements to be made. NOTE: Only ONE replacement is made PER LINE. If the count is left null, it defaults to one. If an asterisk (*) is substituted for the count, the value 65,536 is assumed.

The EDIT command makes full use of all range expressions, qualifier strings, and options; and as noted, a specific parameter field.

The following is an example of a text buffer prior to using an EDIT command, and text buffer after the EDIT command is executed.

```
—△PRINT (CR)
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
—

—△EDIT'LINE',LINE,OF MANY LINES,5 (CR)
—
```

```
--△PRINT ⓒⓡ
THIS IS THE FIRST OF MANY LINES
MARY HAD A LITTLE LAMB
THIS IS THE THIRD OF MANY LINES
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH OF MANY LINES
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH OF MANY LINES
THE LAMB WAS SURE TO GO
THIS IS THE LAST OF MANY LINES
--
```

NOTE: The use of an arbitrary delimiting character permits you to choose a delimiting character not contained in the old or new strings. For example, if a comma (,) is used as a delimiter, it may not appear in either string. If either string contains a comma, you can use a slash (/) as a delimiter.

PRINT

The PRINT command causes the eligible line(s) in the command range to be printed. This is the most common method for viewing portions of the text buffer or the entire text buffer. All forms of the range expressions are utilized. The option commands have no effect. You may use the qualifier string to limit the range expression. There is no parameter field for the PRINT command.

NOTE: While the PRINT command is executing, you may type control characters to aid in viewing the text buffer. See "Character Set," Page 3-3.

BLITZ

The BLITZ command discards all text in the working buffer. Because of the drastic action this command takes, BLITZ followed by a carriage return results in the question "SURE?" A Y in response to SURE? proceeds with BLITZ. You may abort the BLITZ by typing an N or any character except Y. The range option, and qualifier and parameter fields are ignored in a BLITZ command.

USE

The USE command displays the number of lines in the command range, the number of memory bytes currently used, and the number of free bytes. The USE command replies giving three values:

1. A line count. The number of lines within the command range. Type USE to display the total number of lines presently used within the buffer.

2. A byte count. The number of bytes used by the entire working buffer, not simply the lines within the command range.

3. A bytes free count. This is the number of remaining bytes in memory.

You can use the range expression and qualifier strings, but they have little meaning. There is NO parameter field. The following example demonstrates the USE command. The computer employed in this example has 16K of memory with the following text:

```
*       DETERMINE MEMORY LIMIT

INIT1   MOV  M,A    MOVE BYTE
        DAD  D      INCREMENT TRIAL ADDRESS
        MOV  A,M
        DCR  M
        CMP  M
        JNE  INIT1  IF MEMORY CHANGED

INIT2   DCX  H
        SPHL        SET STACK POINTER=MEMORY LIMIT-1
        PUSH H      SET *PC* VALUE ON STACK
        LXI  H,ERROR
        PUSH H      SET:RETURN ADDRESS:

        —
```

NOTE: EDIT requires some room for work space. It refuses to allow more text when there are still 200 free bytes. These 200 bytes are its work space.

```
--△USE ⓒⓡ
LINES = 00015
USED  = 00273
FREE  = 08299
--
```

NEWIN

The NEWIN command opens a disk file for reading. It is not necessary to have an input disk file to do editing. You may create a new file in the buffer by using the INSERT command. The name of the disk file to be opened is contained in the NEWIN command parameter field. The parameter field for the NEWIN command is in the form:

```
<delimiter> <fspec> <delimiter>
```

The default device for <fspec> is SYØ:, there is no default extension.

After the NEWIN command is executed, the file is open for reading. You must use the READ command to read text into the buffer. Note that the NEXT, FLUSH, and BYE commands also cause text to be read. When the end of the file is read, the message "End of File" is typed on the console. EDIT will automatically close the file.

NOTE: If an input file is opened but not read to the end-of-file record, the NEWIN command asks for a go-ahead prior to searching for a new file. The reply Y, to SURE?, instructs NEWIN to proceed to find the new file.

For example.

```
--NEWIN/WORK .TXT/ ⓒⓡ

OLD INPUT FILE NOT FINISHED. ARE YOU SURE?Y ⓒⓡ
```

READ

The READ command is used to input text into the working buffer. Lines are read into the buffer until the end of file is reached, or less than 700 free bytes of buffer space remain. In the first case, EDIT prints the message:

```
END OF FILE
```

In the second case, EDIT prints:

```
NOT ENOUGH RAM
```

Text inputted by the READ command is appended to the working buffer.

NEWOUT

Use the NEWOUT command to open a new output file. The file name is supplied in the parameter field in the same manner as in the NEWIN command. The form of the parameter field is:

```
<delimiter><fspec><delimiter>
```

EDIT will open the file for WRITE, leaving it ready for text. Use the WRITE, FLUSH, or BYE commands, as appropriate, to write text onto the file. Note that any existing file by that name will be overwritten.

The NEWOUT command does not use range expressions, options, or qualifier strings.

If you use NEWOUT without closing a previously opened file, NEWOUT responds with SURE? A reply of Y permits NEWOUT to close the current output file and open the new one. Note that the closing of the previous file will cause any pre-existing file by that name to be replaced by the newly closed one. For example:

```
--NEWOUT/WORK2.TXT/ CR
```

WRITE

The WRITE command outputs text from the buffer onto the output file. It starts at the top of the buffer and continues to the first line of the command range. Thus, the command

```
$WRITE CR
```

writes the entire buffer.

This command uses range expressions, options, and qualifier strings. It has no parameter field. **NOTE: After lines are written, they are deleted from the buffer.**

FLUSH

Use the FLUSH command when editing is complete. It causes the working buffer to be written to the output file, and then any remaining text on the input file is copied over to the output file without modification. The FLUSH command then closes both the input and the output files.

The FLUSH command does not use range expression, options, qualifier strings, or parameter fields. **NOTE: After the lines are written, they are deleted from the buffer.**

NEXT

The NEXT command performs the following sequence:

```
$WRITE
READ
```

This command causes all the text in the buffer to be written to the output file and then refills the buffer from the input file. This command is particularly useful when you are generating large files or when you perform minor editing on large files. The NEXT command does not use a range expression, options, or qualifier strings. It has no parameter field.

BYE

The BYE command is the normal way to exit from EDIT back to the operating system. The BYE command first performs a FLUSH, which causes all remaining text in the buffer and on the input file to be written to the output file. EDIT then closes both input and output files and exits to the operating system.

If no output file has been specified (i.e., no NEWOUT command performed), BYE types the message:

```
NO NEWOUT FILE
```

An output file should be specified via NEWOUT, and the BYE command re-typed. If you wish to exit the editor without writing a file, type CTRL-D.

SEARCH

The SEARCH command scans through a text file for a given character string. The desired character string is specified in the qualifier field. This command begins the scan at the first line in the command range and continues to the end of the buffer (regardless of the last line in the command range). If the given character string is still not found, the buffer is written onto the output file and more text is added from the input file. The SEARCH stops when the end of the file is reached, or when the string is found.

When the string is found, the command range is set to that line. Subsequent commands can reference the line containing the desired string by using an equal (=) for the range expression. The SEARCH command uses the range expression, but does not use option or parameter fields.

For example:

```
--△PRINT (CR)
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTHE LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--


--SEARCH"FIFTHE" (CR)
--=EDITBA,FIFTHE,FIFTH, (CR)
THIS IS THE FIFTHE LINE
THIS IS THE FIFTH LINE
--
```

NOTE: The given character string is enclosed in double quotes ("), not single quotes (').

COMMAND COMPLETION

When EDIT is in the command mode, each terminal keystroke is considered for validity. If the character belongs to no possible command, it is refused and the bell code is echoed to the terminal. If the character is accepted and the command syntax allows only one next character, EDIT supplies and prints this character for you.

In addition to simple syntax checking, EDIT processes command range expressions as they are being entered. If you should enter a range expression referring to nonexistent lines, EDIT refuses further entry and echoes a bell code. Thus, should valid characters be rejected, it indicates that the command range expression is invalid. For example, if you should attempt to type

```
-↑+'TUESDAY'
```

and EDIT refuses the "P" in PRINT, it means that it found no line containing Tuesday. NOTE: This is done before the command was executed, so you can use a backspace to eliminate the TUESDAY and replace this string with a valid string for the text. NOTE: If no information exists in the text buffer, EDIT will not accept commands which need text to be valid. For example, the command PRINT is invalid when no text exists in the text buffer, as there is nothing to print.

APPENDIX A

ERROR MESSAGES

```
<BELL>
```

EDIT rings the terminal's bell when an illegal command character (for the current command) is typed. EDIT also rings the bell if the line expression just entered applies to no line in the text file.

```
FIRST <= LAST
```

The line matching the "first line" expression must occur before, or be the same line as, the line matching the "last line" expression.

```
OLD INPUT FILE NOT FINISHED
ARE YOU SURE?
```

You have attempted to specify a new input file (via NEWIN) before the end-of-file was read on the old one. Replying YES closes the old input file and opens the new one. Replying NO leaves the old input file open.

```
OLD OUTPUT FILE NOT FINISHED
ARE YOU SURE?
```

You have attempted to specify a new output file (via NEWOUT) before the old one was closed. Replying YES closes the old output file and opens the new one. Replying NO leaves the old output file open.

```
ILLEGAL FILE NAME
```

The file specification used in a NEWIN or NEWOUT command contains too many characters. The file specification should contain no blanks or spurious characters.

```
END OF FILE
```

The end-of-file was reached on the input file. The file was automatically closed. This message is informative and does not indicate an error of any kind.

```
NOT FOUND
```

The line specified in the SEARCH command could not be located.

```
NO OUTPUT FILE
```

EDIT could not write the text buffer to disk because no output file has been opened. This message can come in response to a WRITE, FLUSH, SEARCH, NEXT, or BYE command; all of which can cause text to be written to disk.

```
NOT ENOUGH RAM
```

There is not enough free RAM to complete the operation. You can free up RAM by deleting some text lines or by writing some text lines to the output file. Note that this message is a typical response to the READ or NEXT command, and only means that EDIT ran out of RAM space before it read the end-of-file. EDIT automatically stops reading with several hundred bytes still free to allow room for editing.

```
ERROR - xxx ... xxx
```

The operating system detected the error "xxx . . . xxx". The message is normally self-explanatory and is discussed in the appropriate operating system manual appendix.

APPENDIX B

Command Summary

Each of the commands for the Heath Text Editor are summarized in this Appendix. For a detailed explanation of each command, refer to "The Commands," Page 3-12 to 3-22.

Command Structure (Page 3-5)

The general form for an editor command is:

```
[<RANGE EXPRESSION>] [<VERB>] [<QUALIFIER STRING>] [<OPTION>] [<PARAMETERS>]
```

Range Expression Forms (Page 3-5)

1. NULL — First line in previous command range.
2. BLANK — The entire working buffer.
3. = — The previous command range.
4. A single line expression.
5. Multiple expressions.

Line Expression Forms (Pages 3-6 and 3-7)

	EXPRESSION	DEFINITION
1.	↑	The first line in the buffer.
2.	$	The last line in the buffer.
3.	NULL	The first line in the previous command range.
4.	COMMA	Separates multiple line expressions.
5.	+COUNT	Move forward count decimal lines.
6.	−COUNT	Move backward count decimal lines.
7.	+'STRING'	Move forward until 'STRING' is located.
8.	−'STRING'	Move backward until 'STRING' is located.

Verb (Command) Forms

BLITZ	Discards all text after a Y reply to ARE YOU SURE? (Page 3-17).
BYE	Exit to Operating System after flushing text and closing files (Page 3-21).
DELETE	Deletes eligible lines in command range (*except entire buffer) (Page 3-15).
EDIT	Replaces old string with new string once per line. Parameter field: <arbitrary delimiter> old string <arbitrary delimiter> count. Count is decimal number of replacements (Page 3-16).
FLUSH	Writes working buffer, balance of input file, and end-of-file character onto output file. Use when editing is complete. Text is deleted from buffer when complete (Page 3-21).
INSERT	Add to text buffer from keyboard. CTRL-C returns Editor to command mode (Page 3-12).
NEWIN	Opens a new file to be read in. Parameter field specifies file name <delimiter> <name> <delimiter> (Page 3-19).
NEWOUT	Opens a new file for output. Parameter field specifies file name <delimiter><name> <delimiter> (Page 3-20).
NEXT	Writes working buffer onto output file. Then fills buffer from input file (Page 3-21).
PRINT	Print eligible lines on console terminal (Page 3-17).
READ	Reads text into memory from input file.

REPLACE	Replace eligible lines in command range from keyboard. CTRL-C returns Editor to command mode (Page 3-14).
SEARCH "STRING"	Searches text buffer and input file after initial line for 'STRING'. Search stops when STRING is found or end-of-file is found. Command range set to line containing 'STRING' (Page 3-22).
USE	Displays number of lines in buffer, bytes used, and bytes free (Page 3-18).
WRITE	Writes text from the working buffer to output file. Writes start of buffer to first line of common range. After writing, lines are deleted (Page 3-20).

Qualifier String (Page 3-11)

Qualifier string, if present, takes the form 'string'. A qualifier string limits range expression to those lines containing the qualifier string.

Option (Page 3-11)

The option field is:

B	Print line before operating on it.
A	Print line after operating on it.
BA	Print line before and after operating on it.
NULL	No option specified.

Parameter Field (Page 3-12)

This field contains extra parameters needed by the EDIT, NEWIN, and NEWOUT commands. Format is command dependent.

APPENDIX C

The following edited source file is typical of one you might create using EDIT. It is intended to show examples of some of EDIT's editing capabilities.

```
    -- PRINT
               ORG      100000A-160Q
    FPNRM      EQU      073173A
               INX      B           TO
               INX      B            EXPONENT
               LDAX     A
               ANA      B           SET CONDX CODE
               RZ
               DCR      A           /2
               JZ       USRI        IF UNDER FLOW
               DCR      A           /2 again (/4)
    USRI       STAX     B
               CALL     FPNRM
               RET

               END      START

--↑+'EQU'INSERTB
    FPNRM      EQU      073173A
    START      INX      B           INC
--↑+'073173A'EDITBA,073173A,063207A,1
    FPNRM      EQU      073173A
    FPNRM      EQU      063207A
    -- PRINT
               ORG      100000A-160Q
    FPNRM      EQU      063207A
    START      INX      B           INC
               INX      B            TO
               INX      B             EXPONENT
               LDAX     A           (A) = ACCX EXP
               ANA      B           SET CONDX CODE
               RZ
               DCR      A           /2
               JZ       USRI        IF UNDER FLOW
               DCR      A           /2 again (/4)
    USRI       STAX     B
               CALL     FPNRM
               RET

               END      START
```

```
 --↑+'EXPONENT'INSERTB
                 INX      B          EXPONENT
                 LDAX     A          (A)=ACCX EXP
 --'ACC'+1DELETEB
                 LDAX     A
 --↑+'/4'INSERTB
                 DCR      A          /2AGAIN(/4)
      USRI       STAX     B          RET TO ACCX
 --'ACCX'+1DELETEB
      USRI       STAX     B
 --'ACCX'INSERTB
      USRI       STAX     B          RET TO ACCX
                 CALL     FPNRM      NORMALIZE
 --'NORMALIZE'+1DELETEB
                 CALL     FPNRM
 -- PRINT
                 ORG      120000A-160Q
 FPNRM           EQU      063207A
 START           INX      B          INC UP
                 INX      B           TO
                 INX      B            EXPONENT
                 LDAX     B          (A)=ACCX EXP
                 ANA      A          SET CONDX CODE
                 RZ
                 DCR      A          /2
                 JZ       USRI       IF UNDER FLOW
                 DCR      A          /2AGAIN(/4)
 USRI            STAX     B          RET TO ACCX
                 CALL     FPNRM      NORMALIZE
                 RET                 IN CASE 0

                 END      START


--NEWOUT"SY1:TEXT.ASM" ⓒⓡ
--BYE ⓒⓡ
END OF FILE
>                    (HDOS prompt)
```

INDEX