The Yale Editor "E" --
A CRT-Based Text Editing System
(Preliminary Version)

P. Weiner, I. Singh, D. J. Mostow, E. T. Irons

Research Report #19

This work was partially supported by grants from the Alfred P. Sloan Foundation and the Exxon Education Foundation.

April 1973

The Yale Editor "E" -- A CRT-Based Text Editing System

Abstract

Editing programs designed for teletype interaction differ significantly from editing programs designed for CRT interaction. This paper describes the Yale Editor "E," a CRT-based editor, and gives both functional characteristics and implementation details.

I. Introduction

Experience at Yale and elsewhere [1] has shown that low-cost CRT (TV-like) terminals have considerable advantage over any other computer access method. But this is the case only when the CRTs' speed is used properly; use of standard software designed for teletype interaction is sometimes inconvenient on a high speed "teletype equivalent" CRT.

The user of an interactive time-sharing system spends a major part of his time using the system editor. Here too, when teletypes are used for access, it may be better to use an editor designed for teletypes. But our experience shows that editing software designed for use with low-cost high-speed CRTs can provide a capability unattainable with slow speed devices. Indeed, such an editor provides mechanisms for dealing with two-dimensional text in a very natural way. (This is true in spite of the fact that the computer stores text in a compacted linear form.)

This paper describes a CRT-oriented text-editing system implemented on a PDP-10 time-sharing system at Yale University. It functions as a user-mode program under the standard PDP-10 operating system and uses its file structure. The editor has been operational for over three years and it has, we feel, had a significant influence on the way the computer system is used.

II. Hardware

The host system is a typical medium-sized time-sharing system consisting of a PDP-10 computer, disk storage, and other peripheral equipment. Users communicate with the system through Sugarman Laboratories' T-6 Cathode Ray Tube terminals. These are fairly typical of many teletype-compatible CRT terminals currently on the market. Each terminal consists of a TV screen capable of displaying 1600 characters in 20 lines of 80 characters, and a teletype-like keyboard with additional function keys. A basic 64 character font is provided. However, the terminals have an upper/lower case mode in which the upper case characters are distinguished by being displayed in reversed field (black characters on a white background). In addition, characters can also be displayed blinking (protected mode). The terminals are operated in a full duplex mode and each time a key is operated the corresponding ASCII code is transmitted to the computer. Each time a text character is received from the computer, it is written at the cursor position and the cursor is moved to the right. If a control character is received (codes with octal values under 40 represent ASCII control characters), a corresponding terminal control function (e.g., line feed, page clear, etc.) is executed. In normal operation, most characters are echoed back by the teletype service routine in the monitor. The terminal looks to the operating system like a high speed (model 37) teletype.

III. Using the Editor

When a user is editing a file, his CRT screen appears to be a window on the target file. This window can be moved up and down, left or right, or to a different file by operating the appropriate function keys. Simple changes can be made by positioning the cursor at the desired text and overwriting it with the new text. Other functions allow the user to open or close the text, move pieces of text around, and perform various other operations. The cursor is used to point to the place in the file where the editor is to perform an editing function such as inserting or deleting text. In most non-CRT editors, this is accomplished by typing a line number or the surrounding text (context), neither of which is as convenient or natural as being able to point directly using the cursor.

The various editor functions are evoked through the use of function keys that generate ASCII control characters. Since some control characters are not associated with unique keys on the keyboard, many functions are executed by holding down a control-shift key while striking one of the alphanumeric keys. The user can specify parameters with the functions: E.g., when deleting lines, he can specify the number of lines to be deleted. The parameter is preceded by the ENTER key to distinguish it from normal text and is associated with the next function key. For most functions, the system supplies a suitable default

parameter if none is specified, so that pushing a single button is usually sufficient to invoke an editor function.

The editing functions available to the user are described very briefly below. Some of them will be described in more detail in the Appendix.

1. File functions (SET-FILE, MAKE-FILE)

These functions specify the target file. MAKE-FILE is used to create new files, and SET-FILE to examine or edit an existing file.

2. Screen movement functions (+LINES, -LINES, +PAGES, -PAGES, SLIDE-RIGHT, SLIDE-LEFT, +SEARCH, -SEARCH, GOTO)

These functions move the window represented by the CRT screen with respect to the file. +LINES, -LINES, +PAGES, and -PAGES move the window up or down a specified number of lines or pages (a page is defined as 20 lines -- the size of the screen) while SLIDE-LEFT and SLIDE-RIGHT slide the window sideways by a specified number of columns. The two search functions position the window at the n^{th} accurence of a specified key, searching forwards or backwards from the cursor position. The GOTO function is used to position the window at a point that is at a specified percentage of the way through the file. This is commonly used to get to the beginning or the end of the file.

3. Cursor positioning functions (CURSOR-LEFT, CURSOR-RIGHT,

CURSOR-UP, CURSOR-DOWN, RETURN, LINEFEED, HOME, TAB)

These functions move the cursor around within the window without moving the window with respect to the file. They share the property that the response or the echo to the CRT terminal is generated at monitor level by the teletype service routine. This ensures instantaneous response to the user although the editor may not process the functions till somewhat later. The four cursor keys are grouped together on the keyboard and have a built-in auto-repeat which makes it convenient to position the cursor rapidly at any desired place on the screen.

4. Editing functions (INSERT-SPACES, DELETE-SPACES, INSERT-LINES, DELETE-LINES, PICK, PUT)

The first four are used to open or close the text. The PICK and PUT functions are used to copy parts of the file from one place to another, or even to a different file. DELETE-LINES causes an automatic PICK operation to take place. This makes it convenient to move a piece of text from one place to another (rather than copy it). It also allows the user to have second thoughts after doing a DELETE-LINES -- following it by a PUT undoes its effect.

5. Miscellaneous functions (RESET, ENTER, DO-COMMAND, KNOCKDOWN, EXPLAIN, EXIT)

ENTER is used for entering parameters. The DO-COMMAND function

is used to execute frequently used strings of other functions and text. It effectively allows the user to define new functions, using the existing ones. KNOCKDOWN is used to write control characters into the file. EXPLAIN provides expanded descriptions of error messages. RESET can be used to abort a partially specified editing function; it also serves as a general panic button.

The basic way to execute a function is simply to push the button corresponding to the function. However, it is sometimes necessary to specify a parameter to the function such as the name of the file to look at next in the case of SET-FILE. This is accomplished by typing ENTER followed by the parameter and then the function. The ENTER echoes as a blinking ">" to indicate that the system is in parameter mode, and any text typed in goes into a parameter buffer without affecting the file. The parameter mode is terminated when any other function key is typed and the parameters typed in are associated with that function. For entering more than one parameter, the ENTER key is used between successive parameters.

An alternative way to specify a parameter for a function is to move the cursor to a point, hit the ENTER key, move the cursor to another point on the screen, and hit the function key. Thus the user can point out a piece of text to be deleted, or a search key on the screen, instead of typing it in. Unfortunately, since the screen movement keys cannot

be used in parameter mode (because they take parameters themselves), this method of specifying a parameter is limited to within the screen.

Entering a null parameter (ENTER followed by a function key) is distinguished from not entering a parameter at all and it is used to provide a special mode for certain functions. For example, ENTER +LINES is used to initiate scrolling. ENTER +SEARCH specifies the text between the current position and the next space or control character as the search key. This method of specifying a text string on the screen as a parameter is available with the functions +SEARCH, -SEARCH, and SET-FILE. Useful information retrieval aids can be provided to a user by putting appropriate search keys and file names in convenient places. (The system documentation of the PDP-10 is organized this way.)

When a function that takes a parameter is executed without specifying one, the system supplies a default value, which is usually the value last used with the function (or its complement, as explained below). Thus one can do repeated searches against a key, intermixed with other functions, without retyping the key each time. Many of the functions are divided into complementary pairs that share the same default parameters (e.g., +LINES/-LINES, INSERT-SPACES/DELETE-SPACES, etc.). When a parameter is specified for one member of such a pair, it is stored as the default parameter to be used with both functions in the future. This makes it very convenient to undo the effect of one of these functions and get things back the way they were. Thus a +LINES

could be followed by a -LINES or an INSERT-LINES by a DELETE-LINES as long as the second function was executed without an explicit parameter. One can also do things like following an unsuccessful +SEARCH by a -SEARCH without retyping the search key.

An interesting use of the default parameter mechanism, used in our PDP-10 system, comes from initializing the parameter for SET-FILE to a system file called NEWS. The editor is automatically run after a user logs in and the first page of NEWS is presented on the screen, which contains latest system information and a brief description on how to use the system.

IV. Implementation Details

The editor uses the standard file management system provided by the PDP-10 Operating System. The files consist of 128-word (640-character) blocks containing packed ASCII characters, five to the word. The links between the blocks are transparent to a user-mode program; the whole file has to be treated as a contiguous set of blocks. This means that if data has to be inserted or deleted at some point in the middle of the file, the remaining part of the file has to be rewritten. An important design consideration is that a file being edited must survive most system crashes.

A part of the file is kept in a core buffer, where the editing operations are performed. The buffering is transparent to the user, who can move his CRT window freely over the entire file. The file is updated periodically (after every 70 keystrokes) or whenever there is need to access data outside the buffer. This updating involves two operations: First any nulls (ASCII code 0) in the buffer created by the insertion and deletion operations have to be squeezed out, and then, unless the resulting size of the buffer is exactly the same as the corresponding old data on the file, the editor has to ripple down the file reading and rewriting bufferfulls of data so as to move the remaining parts of the file.

Figure 1 shows the mapping between the disk file, the core buffer,

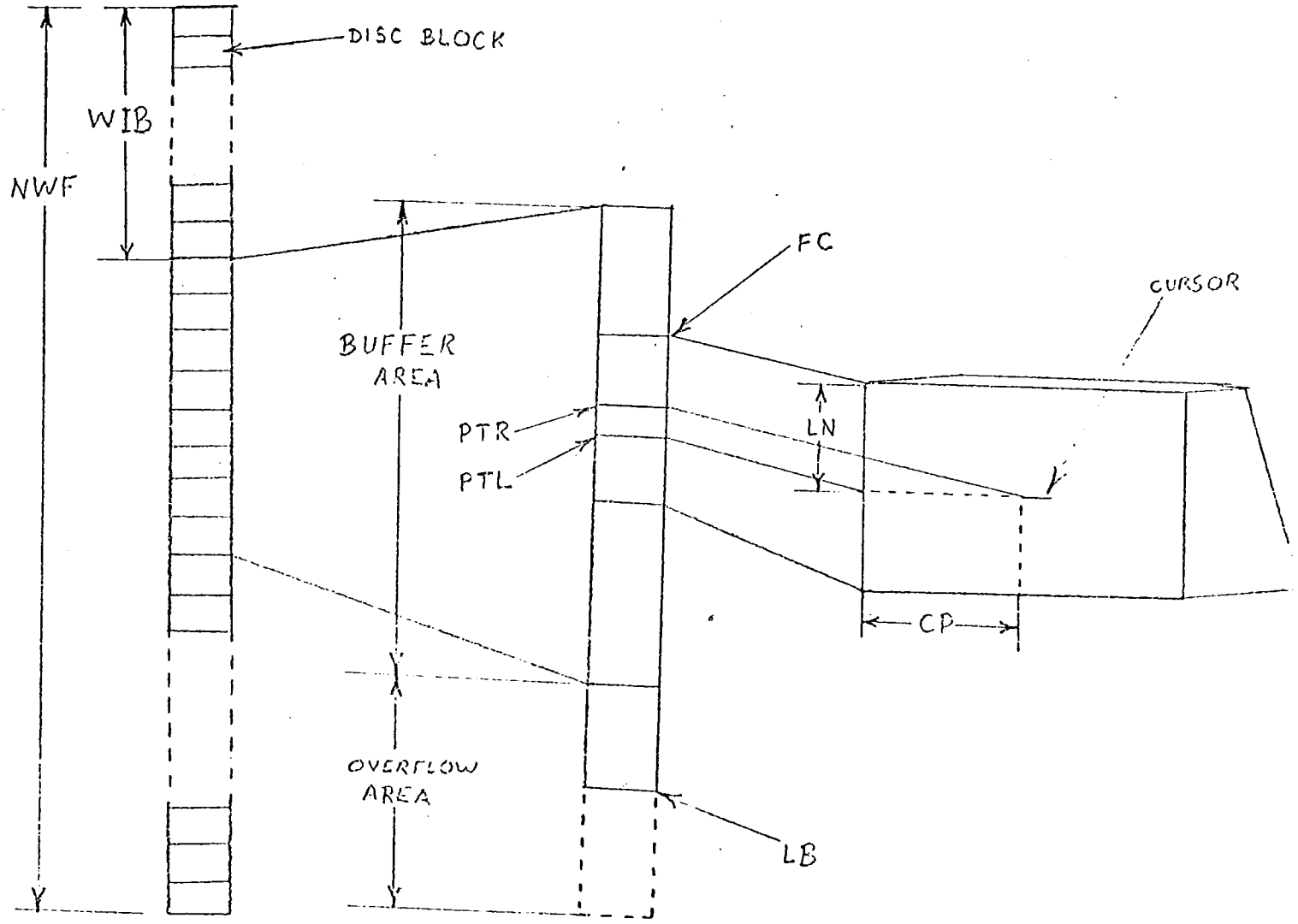
DISC BLOCK
WIB
NWF
BUFFER AREA
FC
CURSOR
PTR
PTL
LN
CP
OVERFLOW AREA
LB
DISC FILE
CORE BUFFER
CRT SCREEN

Figure 1.

and the CRT screen. The labels refer to variables and pointers that are maintained when a file is being edited. These are briefly described below:

NWF Number of words in the file

WIB Address of first word in buffer

PTR Pointer to the character at cursor position

PTL Pointer to the beginning of line containing cursor

FC Pointer to the first character on the screen

LB Pointer to the last character in the buffer

LN Line containing the cursor

CP Column containing the cursor.

The pointers PTR and PTL can be computed using CP, LN, and FC. They are usually recomputed when needed if they have been modified by some previous function. The remaining set of values along with the file directory information, such as the file name, file extension, and the project-programmer number, form the file vector which contains all the information necessary to define the state of the system with respect to the file, including the contents of the screen and the cursor position. The editor maintains two file vectors, one for the current file and another, the old file vector, for the last edited file. When the user types SET-FILE without specifying a parameter the old file vector is used to determine the file and the position of the screen and

the cursor within the file. The current file vector is saved as the old file vector. Thus the user can ping-pong between two files with single keystrokes. This can be quite useful during editing. For example, if large parts of a file under construction already exist in another file, the material can be picked out of that file and deposited in the new file with but a very few button pushes.

When the editing session is terminated by hitting the EXIT key, the two file vectors are stored in an area of core inside the monitor called TMPCOR. When the editor is next run, they are read in so that the target file and the screen are restored to their state at the end of the last session.

The main program modules and the flow of control are outlined in Figure 2. The input handler picks up typed characters and on receiving a control character passes it on to the command interpreter, which is a straightforward dispatch table. Other characters are entered at the current cursor position. When the ENTER key is pushed, the system goes into parameter mode, and received characters are merely put into a parameter buffer.

The Disk I/O Routines perform functions like opening, closing, and updating a file. Here opening a file includes initializing the core buffer and the various pointers in Figure 1. Closing the file includes updating if the data in the buffer has been modified. The updating includes squeezing out nulls and performing the rippling-down

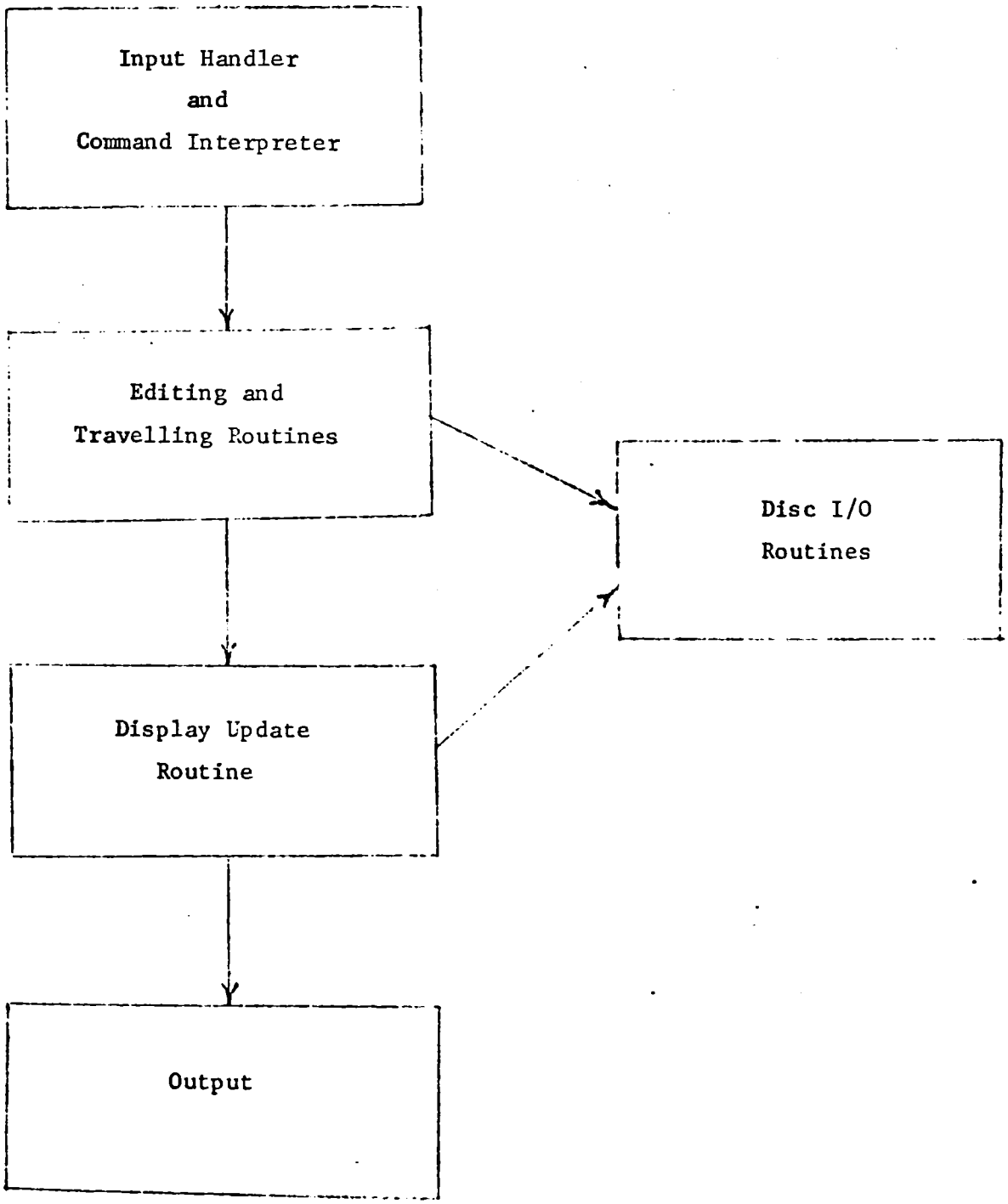
Input Handler
and
Command Interpreter
Editing and
Travelling Routines
Disc I/O
Routines
Display Update
Routine
Output

Figure 2.

operation already described. The relevant pointers have to be suitably updated during the above operations. All transfers between disk and core are done without using a ring of user buffers; data is transmitted between an integral number of logically contiguous blocks and a core buffer area. Random access to any disk block is available through the monitor calls USETI and USETO. The core buffer is moved forwards or backwards by common routines. These routines are called by the various editing, travelling, and display routines when they need to access data outside the core buffer area.

Error messages are handled by a routine that is called by any part of the editor that detects an error. When an error condition occurs, the CRT screen is cleared and a short canned message such as "Search failure" or "Bad parameter" is flashed on the screen. After a few seconds, the previous screen image is restored. For a less cryptic description of the error condition, the user can push the EXPLAIN key. This causes a SET-FILE to be performed to a system file containing a page for each error condition. The appropriate page of this file is displayed on the screen.

There is a group of routines that provide parameter values to routines for functions that take parameters. The calling routine effectively specifies the number and type of parameters that it needs (this is actually accomplished by calling the appropriate subroutine) and the address of the location containing the default value. The

different types of parameters include numbers, strings, and file specifiers (a file specifier looks like DEV:FILENAME.EXT [PROJ.PROG] ⟨PROTECTION⟩, where everything except the file name is optional). The parameter routine picks the parameters up from the parameter buffer, performs the necessary conversions, and fills in default values where possible. For example, in the case of DELETE-LINES, if the user typed in two numbers for the number of lines and characters to delete, these are found in the parameter buffer and returned after ASCII-to-Integer conversion. If only one parameter was typed, the second is set to zero. If the user pointed out the text to be deleted by pushing ENTER and moving the cursor over to a new position, the number of lines and characters to be deleted is computed from the cursor position. Finally, if the user did not specify any parameter at all, a pair of default values is returned.

Insertion and deletion operations take place during the execution of INSERT-LINES, DELETE-LINES, INSERT-SPACES, DELETE-SPACES, and PUT. To insert lines, of course, it is only necessary to insert carriage-return linefeed (CR-LF) pairs. Insertion also takes place when the user types a character beyond the end of a line. In this case an appropriate number of spaces followed by the typed character have to be inserted just before the CR-LF pair that marks the end of the line, so as to make the file identical to what is seen on the screen.

Deletions are performed by overwriting characters in the core

buffer with nulls. These nulls are subsequently squeezed out before the file is updated, unless they are used up by insertion operations. To perform an insertion, the editor searches for nulls up to a certain distance (64 characters) on a character-by-character basis. If enough nulls are found, the remaining non-null characters are pushed down to gather the nulls at the top. Otherwise, the rest of the data in the buffer is moved down a word at a time to make room for the characters to be inserted. Thus an attempt is made to take advantage of nulls created by repeated deletions and insertions. The insertion routine usually inserts a few more nulls than needed in the hope that they can be used in the case of future insertions. In particular, when an insertion is done as a result of typing past the end of a line, several extra nulls are inserted -- there is a very high probability that the user will type in several characters at a time, and one does not want to go through an insertion operation for each one.

The routines that are used by the +SEARCH and -SEARCH functions are also used by +LINES, -LINES, +PAGES, and -PAGES. The first two search for a match against a search key while the last four search for CR-LF pairs. The search is done in a straightforward way: The search key is compared against the buffer contents a character at a time. Upon detecting a mismatch the process is repeated starting the comparison one character further down in the buffer. The calling routine can specify the number of matches to search for and the location of the

search key. The search key can include control characters and wild-card (don't care) characters. The search routines return a pointer to the place in the buffer where the match occurred as well as the number of lines that had to be travelled (CR-LF pairs encountered). This information may be required by the calling routine to determine the way in which the display has to be updated.

Appendix: Detailed Description of the Editor Functions

Some of the editing functions are described in more detail here. Those that have been sufficiently covered are omitted to avoid repetition.

SET-FILE and MAKE-FILE

SET-FILE is used to begin editing a new file. It takes two parameters: a file specifier and a number that specifies the position of the CRT window within the file. If the second parameter is zero, the first page of the file is displayed. A positive number m between 0 and 100 means that the window is to be positioned m percent of the way down the file, while a negative number -m means the window is to be positioned m lines from the beginning of the file. Thus SET-FILE effectively includes a GOTO or +LINES operation.

If the file specifier does not include an extension, and a file with the given name and a null extension does not exist, the editor tries other extensions, such as F4, ALG, MAC, I10, and P11. Finally, if all these fail it repeats the same search in the system file area.

The file specifier can also be picked off the screen by pointing-out the cursor to the beginning of the file name and hitting ENTER and SET-FILE. The parameters for SET-FILE can also be read from a file (indirect specification) by preceding the file name with an "@." Thus if the user types the four keys ENTER @ A SET-FILE, where file A contains

DSKB:LINTRN.DOC[22,40],50, the editor will display a page at the middle (50%) of the file LINTRN.DOC in the user area [22,40] on the DSKB file structure. This indirect look-up can be nested up to a depth of 8.

The initial target file for the editor when it is run can be specified in the run command using a similar format -- R E; DSKB: LINTRN.DOC[22,40],50 or R E; @A.

The MAKE-FILE function is used to create a new file. The file is initialized with a single line:

This is the first line of XYZ

where XYZ is the file name. This line can be edited or deleted.

+LINES, -LINES, +PAGES, -PAGES

These functions move the CRT window up or down by a specified number of lines or pages. +LINES and -LINES share a common parameter, the number of lines to move the window. This is initially set to 7. Similarly +PAGES and -PAGES share a parameter, the number of pages to move, which is initially set to 1. These last two functions simply multiply the number of pages by 20 and transfer control to +LINES or -LINES, which in turn call the search routines to search for the required number of CR-LF pairs.

If the window is moved by more than 20 lines, the display routine is called to rewrite the whole screen. Otherwise, an appropriate number

of ROLL-UP or ROLL-DOWN characters are transmitted to the terminal, and the display routine is instructed to rewrite only the new lines that are to be displayed.

When any of these functions are executed with a null parameter (e.g., ENTER +LINES), a scrolling mode is initiated. In this mode, the editor continuously repeats the specified operations (+LINES in this case). The speed of the scrolling can be controlled by hitting different numeric keys. Hitting a "-" (minus) changes the direction of the scrolling. The effect is somewhat like operating a microfilm viewer in that the user can move the file past the screen at variable speeds in either direction.

+SEARCH and -SEARCH

These two functions search forwards or backwards from the current position as defined by the cursor for the n^{th} occurence of a search key. They take two parameters: a character string for a search key, and a number that specifies how many matches to search for. The search string may contain control characters which can be entered by using the KNOCKDOWN key and wild-card characters entered by typing KNOCKDOWN ?.

The search key can also be picked up from the screen (i.e., the current target file) in the usual way by pointing-out with the cursor. If the search key is a single word (or a string not containing any embedded spaces) the user can simply move the cursor to the start of

the word and hit ENTER +SEARCH or -SEARCH.

In the event of an unsuccessful search, the error message "Search failure" is flashed and the screen is restored. Since the editor has to search up to the end (or the beginning) of the file and this involves moving the core buffer with respect to the file, the current file vector is saved at the beginning of a search operation and used to restore things in the event of a failure.

If a successful match is found within the text on the screen, the cursor is positioned to point to it. Otherwise, the CRT window is moved so that the first line contains the string that was matched and the cursor is left pointing to it.

PICK and PUT

These functions, along with DELETE-LINES (which automatically executes a pick operation) allow pieces of text to be moved around between different parts of the file or between different files. The text to be picked up can be specified numerically by two numbers -- the number of lines and characters starting from the current position. Alternatively the user may point it out with the cursor. The text thus specified is copied into a put buffer one character at a time. If it is too long for the put buffer, which can hold only 1280 characters, the put buffer is cleared and a temporary put file is created in the user's area with the name "nnnPUT.TMP" where nnn is the user's job

number.

The PUT function inserts the contents of the put buffer at the current location. If the put buffer is empty, it tries to use the put file and if that cannot be found an error message results.

DO-COMMAND

The DO-COMMAND function provides an open-ended extension mechanism: The user can define new functions utilizing other editor functions in a very convenient way. This function effectively performs a delayed interpretation of button pushes. The editor is put in a mode where it picks up characters from a command buffer instead of receiving them from the keyboard and interprets them in the usual way. The command string consisting of the sequence of characters to be interpreted can be entered into the command buffer in one of the following ways:

a) It may be explicitly typed in as the first parameter to DO-COMMAND using the KNOCKDOWN function to enter the control characters.

b) It can be read in from a file (indirect mode) by entering the file name with a "@" ahead of it as the first parameter (e.g., ENTER @C DO-COMMAND where the file C contains the command string).

c) The editor can create the command string by observing

the user execute the desired sequence of operations. The editor is put into a learn mode by typing ENTER DO-COMMAND (i.e., DO-COMMAND with a null parameter). All subsequent button pushes are recorded in the command buffer besides being normally interpreted. The mode is terminated with another push of the DO-COMMAND button.

The second parameter to DO-COMMAND specifies the number of times the command string is to be interpreted (or executed). Either of these two parameters may be omitted in which case the default values are the last ones used.

A third parameter, a file specifier, allows the contents of the command buffer to be written out on a file. Thus the user can create a command string using the learn mode, save it on a file, and call it when required, using the indirect mode.

The display routines are inactivated during a DO-COMMAND operation. Other than that the characters received from the command buffer are treated in exactly the same way as if they were received from the keyboard. The command string execution is terminated when it has been executed the specified number of times and then the whole screen is rewritten. Any error condition also terminates the operation.

Reference

Edgar T. Irons and Frans M. Djorup. A CRT editing system. Communications of the ACM, volume 15, number 1, January 1972, 16-20.