

33-8111-0714
RETURN TO LIBRARY

ARPA ORDER NO.: 189-1
7P10 Information Processing Techniques

R-2000-ARPA
July 1977

A Guide to NED: A New On-Line Computer Editor

Jeanne Kelly

A report prepared for
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

Rand
SANTA MONICA, CA. 90406

The research described in this report was sponsored by the Defense Advanced Research Projects Agency under Contract No. DAHC15-73-C-0181.

Reports of The Rand Corporation do not necessarily reflect the opinions or policies of the sponsors of Rand research.

R-2000-ARPA
July 1977

A Guide to NED: A New On-Line Computer Editor

Jeanne Kelly

A report prepared for
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY



PREFACE

This report is a detailed, step-by-step guide for using the New Editor (NED), a computer program designed to create and modify text from a cathode ray tube (CRT) terminal. The work was funded by the Defense Advanced Research Projects Agency (ARPA).

The New Editor was written in the systems programming language C. It is accessed using modified Ann Arbor Model K4080 D terminals, which are supported by UNIX, a timesharing operating system designed and implemented by Bell Laboratories. UNIX operates on the PDP 11-40/45/70 series of Digital Equipment Corporation minicomputers. (For further information on UNIX, refer to "Documents for Use with the UNIX Timesharing System," Bell Telephone Laboratories, obtainable from the Western Electric Company.)

Both those who want to use NED and those who are simply interested in understanding some of its capabilities should find this guide informative and easy to follow, even without the use of the computer. Users, however, must have on-line access to a PDP-11 via an Ann Arbor terminal because of the visual features of the New Editor. Extensive computer experience is not necessary in order to use this guide, although some computer background is helpful.

At the time of publication, this is the only formal documentation available about the New Editor. The New Editor is being given limited distribution to members of the ARPA research community for testing and evaluation.

SUMMARY

The New Editor (NED) is a computer program designed to help individuals create and modify text on a cathode ray tube terminal, taking full advantage of the features unique to a CRT, with its screen, cursor, and keyboard. By integrating the Editor's software with the CRT's hardware, a user-oriented editing system has been developed that is convenient and easy to work with.

This report explains how to use the current version of NED but does not attempt to list all of its capabilities, many of which are still being extended. With the background given here, users should be able, however, to proceed and develop further use of NED on their own.

Part 1 of the report supplies general information on using the Editor and indicates the types of operations that the Editor can perform.

Part 2 is a step-by-step demonstration of the Editor. It shows how the Editor takes advantage of the CRT's screen, keyboard, and cursor. Editing is not restricted to working from top to bottom or left to right; a user can edit in any direction at any place on the screen. More than one file (a file is a collection of text) and/or different parts of the same file, may be viewed and edited on the screen concurrently by dividing the screen into different sections or windows. Keyboard buttons (keys) are used to edit text viewed on the CRT screen. In addition to the standard typewriter keys, special keys have been programmed to initiate operations for faster and more convenient editing. The cursor, an underscore displayed on the CRT screen, indicates where editing operations are to take place. It is controlled by some of the special keys on the the terminal's keyboard and can be moved readily to any location on the screen.

Editing is explained in detail, starting with logging on to the computer, using the Editor, and terminating the session. An example of editing text filled with typical errors is included to illustrate Editor usage. If followed closely, this guide is sufficient to enable the user to learn some of the Editor's special features as well as basic text editing.

ACKNOWLEDGMENTS

I would like to thank my reviewers for their help in writing this report. In particular, I would like to express my appreciation to Ann Valsasina for her patience and for the many important contributions she made to this report.

Toni Shetler and Clara Lai also deserve special thanks for their helpful comments and suggestions.

CONTENTS

PREFACE.....	iii
SUMMARY	v
ACKNOWLEDGMENTS	vii

PART 1. THE NEW EDITOR

I. PRELIMINARY INFORMATION	3
II. INVOKING THE EDITOR AND USING THE EDITOR BUTTONS..	7
III. EDITOR COMMAND FORMAT.....	12
IV. SUMMARY OF EDITOR COMMANDS.....	14

PART 2. THE DEMONSTRATION

V. THE DEMONSTRATION FORMAT AND LOG IN/OUT	19
VI. BASIC EDITOR CONCEPTS	21
A. Invoking the Editor.....	21
B. Cursor Motion Buttons and Tabs	22
C. Editor Command Format Examples.....	23
D. Window Motion	23
E. Basic Editing and Text Creation	24
VII. MORE EDITOR CONCEPTS	27
A. Calling a File.....	27
B. String Searching.....	28
C. Adding and Deleting Lines	28
D. Pick and Put	29
E. Cursor Motion Button Arguments	31
VIII. MULTIPLE WINDOWS.....	33
IX. TERMINATION.....	35
X. AN EXAMPLE OF EDITING "ALICE".....	36
XI. RECOVERY.....	39

PART 1. THE NEW EDITOR

I. PRELIMINARY INFORMATION

The New Editor is an interactive computer program for creating and modifying *text*. Text is a collection of lines of information containing characters consisting of upper and lower case letters, numbers, punctuation marks, and a few special symbols. A series of characters is referred to as a *string*.

Text entered into the computer's memory, by way of the Editor or by some other means, is stored as a unit called a *file*. Usually a file is a collection of related information such as a document, a computer program, a data file, or a series of records. Each file is given a file name to be specified when saving or recalling the file. The Editor can be used to enter text into a computer and store it as a file. Modifications can be made easily by recalling a file and modifying the existing text through use of the Editor.

The Editor views files through a *window*, which is a rectangular area on the CRT screen. Files are defined in terms of lines, partial lines, columns, and areas. Files have the same basic structure, whether they are created using the Editor or by other methods.

Example:

```
This is a line of text.  
This is another line of text.
```

In this example, there are three lines of text. The first line contains 23 characters (spaces and punctuation are included), the second line is a null line containing zero characters, and the third line contains 29 characters. Each line starts at the left margin (perhaps beginning with a space or spaces), and can extend indefinitely to the right. A partial line is a section of a line that starts and ends with a specified character. In the above example, the text "another line" is a partial line.

The Editor can refer to each line in the file by line number, although these numbers do not actually appear in the text.

Example:

line number	column number
	123456789.....
1	This is a line of text.
2	
3	This is another line of text.

A file can be thought of in terms of a grid or matrix containing horizontal lines and vertical columns of characters. A column is a vertical series of contiguous characters which can be one character position within a line, a series of several characters within a column, or all the characters from the first to last line of the file that are in the same position in each line.

A rectangular section of text is called an *area*. In this example, the enclosed characters compose an area of text.

Example:

```
This is a line of text.
This is another line of text.
This is more text.
```

An area of text is a series of contiguous partial lines, each partial line starting and ending in the same column.

The Editor knows how a file is structured and accesses it by the file name. The file is viewed through the formatted window that the Editor displays on the screen. Figure 1 illustrates the Editor's format on the CRT screen.

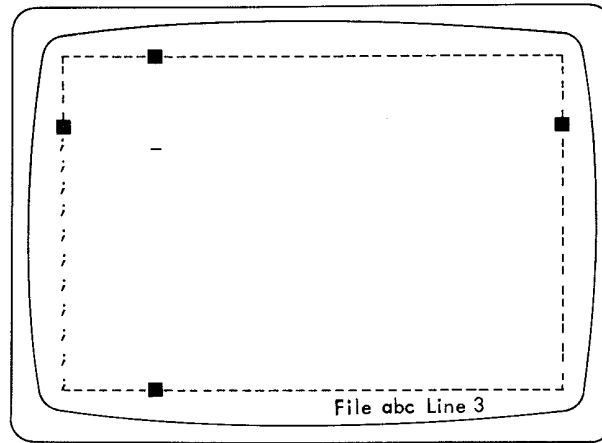


Fig. 1—Editor format on CRT screen

The CRT screen is 80 columns wide and 40 lines long. The Editor window format uses three lines and two columns, leaving a window 78 columns by 37 lines. Thus a section of the file 78 columns by 37 lines may be viewed at one time. The number of lines in a window composes a *page*. If there is one window on the screen, one page equals 37 lines. The rest of the file can be viewed by *paging* or moving the window forward or backward. Text stored past column 78 can be viewed by moving the window to the right.

Figure 2 shows how a file would appear through the Editor window. This text was stored under the file name "abc" as displayed on the bottom line of the screen. The number 3 is the number of the line currently containing the cursor, an underscore displayed on the CRT screen to indicate position. The cursor, which can be easily moved using *cursor positioning keys*, is used for indicating where text is to be added, deleted, or modified. To help identify the cursor position, there are markers on each border of the Editor window.

The screen initially contains one window but can be divided into as many as 10 windows. Each of these windows can view a different file or a different part of the same file.

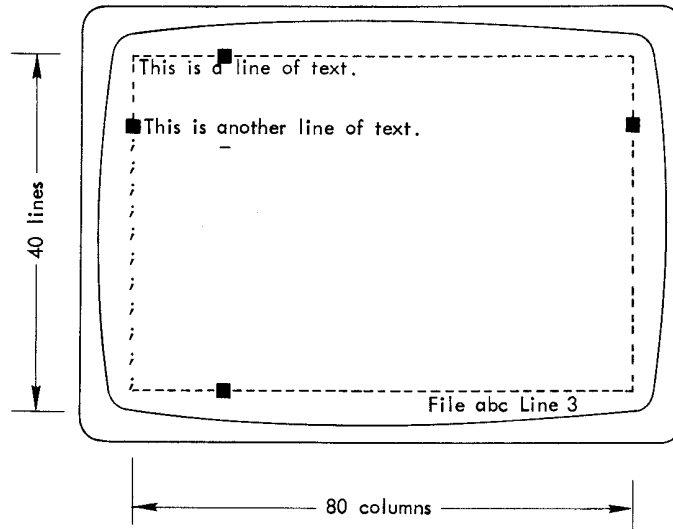


Fig. 2—Editor format and text

In Fig. 3, there are three windows on the CRT screen. The active window is the one currently containing the cursor and having dominant window boundaries. The message at the bottom of the screen refers to the file and line number of the active window. The three files viewed on the screen are the current files. If another file

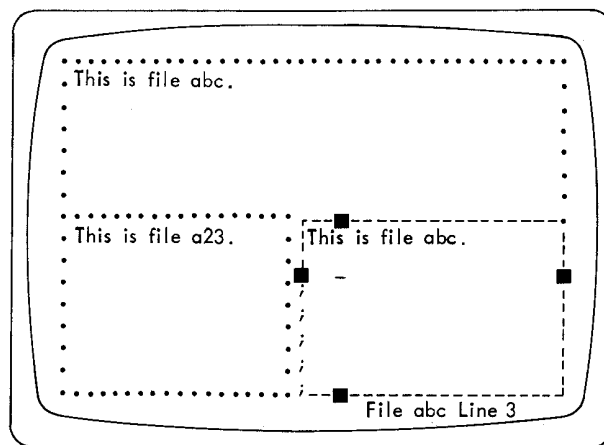


Fig. 3—Editor format and windows

were brought into one of the windows for viewing, the current file would become the alternate file, and the new file would become the current file.

Having been introduced to the Editor and some of the necessary terminology in Section 1, the user can now start the demonstrations in Part 2, using the computer terminal. The remainder of Part 1 of this guide (Sections II through IV), however, contains additional introductory material about the Editor and its capabilities, and should be read before Part 2, especially by those new to text editing or the New Editor, and by those not using a computer terminal in conjunction with Part 2.

Using a computer terminal with access to the Editor, while not necessary, will aid users in understanding Editor usage in Part 2 of the guide (Sections V through XI). Section V explains the demonstration format and logging on to the computer. Sections VI through VIII contain demonstrations using the Editor functions. Section IX gives termination procedures for logging off the computer, and Section X provides an example of text editing using the information from the other sections. Finally, Section XI shows the user how to restore files in case of Editor malfunction.

Extra copies of the keyboard diagram (Fig. 4 on p. 8) and a summary of the Editor commands (Section IV), may be found in the pocket inside the back cover of this report. These can serve as reference aids while the Editor is being used.

II. INVOKING THE EDITOR AND USING THE EDITOR BUTTONS

After the user logs on to the computer, the computer's operating system, UNIX, prompts the user with a **%** prompt character displayed on the CRT screen, indicating that the terminal is ready to accept a command. The user invokes the New Editor by typing the command *ned* in response to the UNIX prompt character **%**. An Editor function identifies the operations that the Editor performs. These functions are invoked by pushing the button on the keyboard labeled according to a particular Editor function. Using the individual Editor function buttons allows for basic editing capabilities. These function buttons are actually a simplified version of an Editor command, explained in the following section, which allows for more extensive editing capabilities.

The cursor is controlled by cursor motion buttons that designate where operations are to be performed by the Editor functions. The cursor can be used to indicate the position for making changes not requiring the use of the Editor functions, as in the case of overtyping.

The Editor operates in either INSERT MODE or OVERTYPE MODE, but is initially in OVERTYPE MODE. The INSERT MODE button switches the Editor between modes. While in the INSERT MODE, characters may be inserted or deleted; in OVERTYPE MODE, they may be replaced or deleted.

Most of the function buttons and cursor motion buttons are located among the special keys to the right of the standard keyboard. Other Editor buttons are located in the standard keyboard. (See Fig. 4.) For reading convenience, the buttons described by more than one character are enclosed to distinguish them from single character buttons:



The enclosed PUT means to press the PUT button.

PUT

The letters *PUT* mean to type the upper case letters *P*, then *U*, followed by a *T*.

The keyboard diagram (Fig. 4) shows the location of the following Editor keys. The characters indicating the Editor function appear either on the top or on the front of the keys.



Move forward through the file by one or more lines.



Move backward through the file by one or more lines.



Move forward through the file by paging. (One page equals the number of lines in the current window—usually 37.)

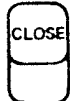
Fig. 4—Keyboard diagram



Move backward through the file by paging.



Go to an identified location in the file.



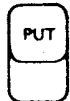
Remove information from the file.



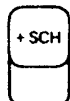
Open blank space within the file.



Pick out specified information that can then be PUT somewhere else.



Add previously PICKed information to the current file.



Search forward in the file for a specified string.



Search backward in the file for a specified string.



Exit from the Editor.

The following Editor functions are used while holding down the CTRL button, i.e., the CTRL button must be depressed and held before depressing the Editor function button. The CTRL button provides an additional case shift key; it allows ordinary keyboard keys to be used as function buttons. The shaded buttons indicate that the CTRL button is used in conjunction with these Editor keys.



Move the window to the left.



Move the window to the right.



Save the file being viewed.



Call a file into the Editor for editing.



Create a new window or remove a window.



Move the cursor to another window.



Set or reset tab stops.

The INSERT MODE button is not an Editor function but a mode of operation. The DEL CHAR button deletes a character at the cursor position.



This button switches the Editor between the INSERT MODE and the OVERTYPE MODE, alternating modes with each press of the button.



This button deletes a character. The remaining characters in the line move to the left, closing up the space.

The cursor has several buttons to control its movement. These are called *cursor motion buttons* (CMB) and are described briefly below. (Also see Fig. 4.) The directional cursor control buttons cause the cursor to begin moving when you press the button and to stop when you stop pressing the button.



Move cursor down.



Move cursor up.



Move cursor to the right.



Move cursor to the left.



Move cursor to upper left-hand corner of the window.



Move cursor to beginning of the next line.



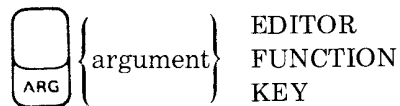
Go to the next forward tab stop.



Back up to the previous tab stop.

III. EDITOR COMMAND FORMAT

Many editing operations can be done simply by pushing the appropriate Editor function button. For more sophisticated editing, there is a standardized Editor command format. In many instances, the Editor function buttons are a simplified version of this format. The Editor command format is a combination of the ARG button, an argument, and one of the Editor function buttons (keys):



argument

The ARG button indicates that editing information, or an argument, is to be passed to the Editor function.

An argument is editing information used by an Editor function to help define the actual editing operation. There are three types of arguments: numeric strings, character strings, and cursor motion buttons.

numeric string

An integer, which may be preceded by a minus sign.

character string

A series of alphabetic or numeric characters. (Note: Not meaningful to all functions.)

cursor motion
buttons

A sequence of cursor motion buttons. (Note: Not meaningful to all functions.)

Common sense can usually tell you when a certain type of argument is not appropriate, but as a safeguard, error messages are displayed. This demonstration will specify the correct kind of argument to avoid confusion.

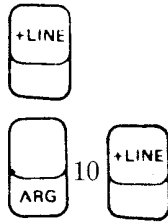
EDITOR
FUNCTION
KEY

An Editor function key indicates the type of operation that the Editor is to perform.

The Editor command format is applicable to all of the Editor functions except PUT, S/R TAB, and CH WIN, which do not have any defined arguments. All of the other functions accept one or more argument types.

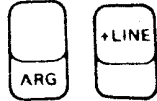
There are two special cases of the Editor command format: using the Editor function by itself (without an intervening argument), and using the ARG button followed by an Editor function button.

Using the Editor function by itself, in most instances, is a simplified version of the Editor command format. Consider the following examples:



Using the +LINE Editor function button causes the Editor to move the window forward one fourth of a page, or 10 lines for a normal sized window. Using the complete format in the second example would have the same effect.

In the second special case of the Editor command format, when the ARG button is used followed by an Editor function button, there is a predefined Editor operation for the function. Not all of the functions have this case defined. (See Section IV.) Sometimes the predefined Editor operation means to initiate a previously defined argument, as in the following example. Consider again the +LINE function:



Preceding the +LINE button by the ARG button will cause the line at which the cursor is currently positioned to be moved to the top of the window. This same effect can be attained by using the full format and figuring out exactly how many lines to move the window forward.

In other instances, there is a specifically defined operation associated with this special case of the Editor command format, as in the following example:



This sequence removes the last window created. There is not a predefined argument in the above example, but there is a predefined operation.

IV. SUMMARY OF EDITOR COMMANDS

KEY	EDITOR FUNCTION	ARG argument EDITOR FUNCTION (Argument type)			ARG EDITOR FUNCTION
		Numeric (N)	Character String (C)	Cursor Motion Buttons	
GOTO	Aligns top of window and first line of file	Moves window to view line N of file	NA	NA	Moves window to view last line of file
+LINE - LINE	Moves window ¼ page forward (+) or backward (-)	Moves window N lines forward (+) or backward (-)	NA	NA	Puts CCL at top (+) or bottom (-) of window
LEFT	Moves window to left 16 columns or to file boundary, whichever comes first	Moves window N columns to left in file	NA	NA	NA
RIGHT	Moves window 16 columns to right in file	Moves window N columns to right in file	NA	NA	NA
+PAGE - PAGE	Moves window forward (+) or backward (-) 1 page	Moves window forward (+) or backward (-) N pages	NA	NA	NA
+SCH -SCH	Searches for previous search string, if any	N is treated like C	Searches forward (+) or backward (-) in file for C	NA	Searches for string pointed to by cursor until first blank
OPEN	Inserts blank line above CCL	Inserts N blank lines above CCL	NA	Inserts blank partial lines, lines, or areas defined by cursor	Moves partial line to right of current cursor position to a new next line
CLOSE	Deletes CCL and puts in CLOSE buffer	Deletes CCL plus N-1 successive lines; places deleted lines in CLOSE buffer	NA	Deletes partial lines, lines, or areas defined by cursor	Deletes partial line to right of cursor replacing it with next line
PICK	Places CCL in PICK buffer	Places CCL plus N-1 successive lines in PICK buffer	NA	Places partial lines, lines, or areas defined by cursor in PICK buffer	NA
PUT	Places contents of PICK buffer above CCL	NA	NA	NA	Places contents of CLOSE buffer above CCL

Summary of Editor Commands (continued)

KEY	EDITOR FUNCTION	ARG argument EDITOR FUNCTION (Argument type)			ARG EDITOR FUNCTION
		Numeric (N)	Character String (C)	Cursor Motion Buttons	
WIN	Makes a new window whose boundary is defined by the cursor position in line or column 1	N is treated like C	Makes a new window containing a file named C	NA	Removes last window created
CH WIN	Alternates cursor sequentially through windows	NA	NA	NA	Moves cursor to original window
USE	Switches from current file to alternate file; if no alternate file, uses last previously edited file	N is treated like C	File C becomes current file; previous file becomes alternate file	NA	String pointed to by cursor (up to first blank) becomes current file
SAVE	Writes current file on disk under existing file name	N is treated like C	Writes current file on disk using C as file name	NA	Treats string pointed to by cursor (up to first blank) like C
EXIT	Exits Editor, writing all files on disk; old versions of files become "filename.bak" files	Exits Editor	If C = a, exits without writing files—no modifications will be applied to original files; otherwise exits Editor	Exits Editor	Exits Editor
S/R TAB	Sets tab stop at current cursor position	NA	NA	NA	Removes tab stop from current cursor position

NA = not applicable.
CCL = current cursor line.

PART 2. THE DEMONSTRATION

V. THE DEMONSTRATION FORMAT AND LOG IN/OUT

The demonstrations in the following three sections have a standardized description format. An introductory explanation is followed by a page that is divided. The left-hand side of the page indicates which buttons the user should push, and the right-hand side presents explanatory comments. To follow through the demonstrations, push all the buttons, except those in boldface type, indicated on the left side of the page. The characters in boldface type are displayed by the computer; no boldface type will appear on the screen, however. The demonstration is in several sections so that it can be done during different sessions. Simple directions are given for beginning and ending each of the demonstrations. If there are any problems with the Editor, see Section XI on recovery.

Section IX is an example of editing text filled with typical errors. The demonstration indicates which editor functions to use to correct these errors. Try this section after becoming familiar with the editor functions in the preceding sections.

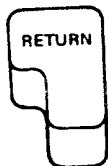
Each section begins with logging on to the computer and ends with logging out. The UNIX operating system provides a standard procedure for log in and log out, which can be modified to suit a particular installation's needs. The Rand Corporation modified the log in procedure and uses the standard log out procedure. Check with your installation administrator to obtain your log in name/account and to find out if the log in/out procedure has been modified. Below are examples of the standard UNIX log in/out procedure and of The Rand Corporation's modified log in procedure.

This is the standard UNIX log in procedure:



Depress the CTRL button and the letter *d* to initiate log on.

;**LOGIN:** name

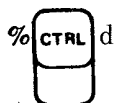


The boldface characters are output by UNIX. **;**LOGIN:**** indicates that UNIX is ready to accept your log in name. Type in your log in name, instead of *name*, and then press the RETURN button. If you make a mistake, an error message will be displayed, and you will be prompted to retype your log in name. A system message may appear on the screen: read it and continue.

%

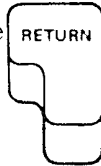
A percent sign appears on the screen indicating that the terminal is ready to accept input.

This is the standard UNIX log out procedure:




Depress the CTRL button and the letter *d* to log out from UNIX.

This is Rand's modified log in procedure:

User:name 

The boldface characters are output by UNIX. The prompt **User:** should appear on the screen indicating that UNIX is ready to accept your log in name. Even if this prompt does not appear, type in your UNIX log in name instead of *name* and then push the RETURN button. If you make a mistake, an error message will be displayed, and you will be prompted to retype your log in name.

Password:p12 

Type in your password instead of *p12*. Your password is set to the first three characters of your employee number unless you have defined it differently. Your password, although typed, will not appear on the screen for security reasons. A system message may appear on the screen: read it and continue.

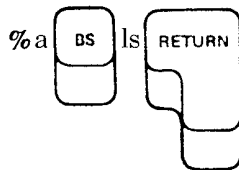
%

A percent sign appears on the screen indicating that the terminal is ready to accept input.

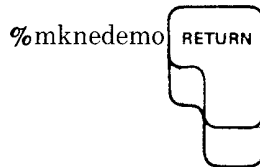
VI. BASIC EDITOR CONCEPTS

PART A. INVOKING THE EDITOR

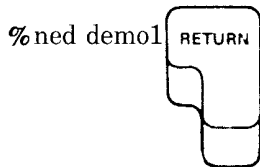
Log in to the computer as illustrated in Section V.



% indicates that UNIX is ready to accept commands. Typing errors can be corrected by using the BS button to erase mistakes. In this example, an *a* was typed in error. *ls* means to list the names of any files you have that are known to UNIX. These names are kept in your directory.



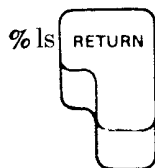
This special UNIX command copies files *demo1*, *demo2*, and *demo.alice* into your directory. If these file names already exist, the files will not be copied. The existing file names must be changed or the files deleted in order to copy the demonstration files into your account.



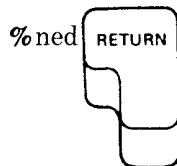
ned calls the New Editor to edit file *demo1*; at this point the Editor is invoked.



At any time you may discontinue the demonstration by pushing the EXIT button to return to UNIX. If you have made any changes to your file, the file with its latest modifications will automatically be saved in place of the original file. (Note: the most recent version of the edited file is also saved as a backup.)



This UNIX command lists your directory. *demo1*, *demo2*, and *demo.alice* should appear in your directory.



Invokes *ned* to edit your file *demo1*. Using *ned* without a file name will cause the Editor to call the last file viewed by the Editor.

This ends Part A. To terminate, see Section IX; otherwise do Part B.

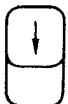
PART B. CURSOR MOTION BUTTONS AND TABS

Part B is designed to familiarize you with the cursor motion buttons. It is important to know how to use these buttons because the cursor position indicates where editing takes place. Each side of the screen contains a marker indicating where the cursor is positioned. Try using all the buttons as indicated in the following examples.

On the bottom of the screen is the message: **File demo1 Line 11:** demo1 is the file being edited, and 11 is the current cursor line number within the file.



Moves the cursor one space to the right. If you hold this button down, the cursor will wrap around the screen; i.e., the cursor will move to the beginning of the current line when it reaches the end.



Moves the cursor down one line at a time and wraps around when it reaches the bottom of the screen.



Moves the cursor up one line at a time, and similarly wraps around when it reaches the top.



Moves the cursor to the left and wraps around when it reaches the beginning of the line.



Returns the cursor to the upper left position of the window.



Positions the cursor in column 3 when this button is pushed twice.



Pushing the CTRL button and holding it down while pushing the S/R TAB button will cause a tab to be set in column 3 (or wherever the cursor is positioned).



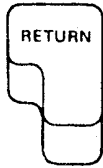
Pushing the +TAB button will move the cursor to the next tab position. There are default tab positions every 8 columns.



Pushing the -TAB button causes the cursor to back up to last tab stop. The cursor should now be in column 3.



This sequence will remove the tab stop located by the cursor.



The RETURN button causes the cursor to move to the beginning of the next line.



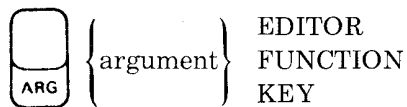
You should be positioned at the first default tab stop in column 8.



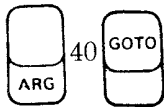
The HOME button returns the cursor to the upper left-hand corner of the window.

PART C. EDITOR COMMAND FORMAT EXAMPLES

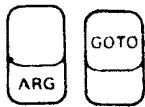
As discussed in Section III, the basic Editor command format is:



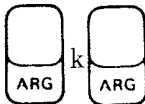
The ARG button and argument are not always required because predefined arguments or operations have been established for editing convenience. To help clarify this concept, try the GOTO function as demonstrated below in three different ways.



ARG indicates that an argument is to be passed to the GOTO function. The GOTO function receives the argument 40 which causes the Editor to position line 40 one fourth of the way down the window.



This is a special case of the GOTO function. The last line of the file is placed one fourth of the way from the top of the window.



Using the ARG button a second time nullifies the original ARG, and the argument, if any.



The other special case of the GOTO function positions the first line of the file at the top of the window.

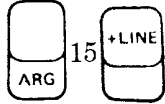
PART D. WINDOW MOTION

To conveniently scan through your file, several additional functions have been defined. Note that +, or forward, in a file means advancing toward the end of your

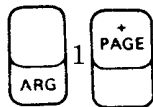
file and that —, or backward, means moving toward the beginning of the file. Also, the number of lines in a page depends on window size. Usually one page, or a full screen, is equivalent to 37 lines (i.e., a full CRT screen window). The file is stationary, while the window moves to view different parts of the file.



Move the window 1 page forward, displaying the next page of your file.



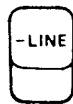
Move the window 15 lines forward.



Move the window ahead another page (same effect as the + PAGE function button). If 10 were used instead of 1, the window would move ahead 10 pages.



Move the window back 1 page.



Move the window back 10 lines (or approximately the number of lines in the window divided by 4).



Move the window to the right 16 columns. (16 is the default number of columns—an argument other than 16 may be specified). Right refers to the window's movement.



Move the window to the left 16 columns, or to the file boundary—whichever comes first.



Return the window to the beginning of your file.

PART E. BASIC EDITING AND TEXT CREATION

There are two modes of editing: OVERTYPE MODE and INSERT MODE. Overtyping means typing over the current text in your file, thereby replacing it. The cursor is positioned where the change is to be made or new text is to be added, and then the correction or addition is simply typed onto the screen. Initially the Editor is in the OVERTYPE MODE.

INSERT MODE, which refers to adding or deleting characters within a line, is invoked by pushing the INSERT MODE button. (Pushing the button again returns the user to OVERTYPE MODE.) The cursor designates where characters are to be added or deleted. The characters to the right of the new insertion are moved to the

right as new characters are added. As characters are deleted, the characters to the right of the deletion are moved to the left. INSERT MODE and OVERTYPE MODE have the same result when adding text to a blank line or when there is no previous text to the right of the new text.

Some very basic illustrations are presented below:



The cursor is positioned at the left-hand corner of the window.



Move the cursor down to line 3 by pushing this button twice.



Push the +TAB button. (You should be in column 8.)

This is new text.

Type this sentence on the keyboard. This is an example of adding new text in the OVERTYPE MODE.



RETURN returns you to the next line.



Back up to the previous line by pushing this button once, moving the cursor to line 3.



Press the +TAB button.



Move the cursor to the right until it is under the *n* in *new*.

overtyping.

Type *overtyping*. over *new text*. (This illustrates typing over existing text.)



Move the cursor to the left until it is positioned at the *o* in *overtyping*.



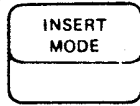
Press this button 10 times. The word *overtyping* is erased, and the line now says, *This is*.



Press the INSERT MODE button. (You are now in INSERT MODE.)

insert mode

Now, type in *insert mode*. Notice that the characters are inserted between the s in *is* and the . (period).

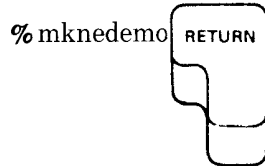


Press this button to leave INSERT MODE and return to OVERTYPE MODE.

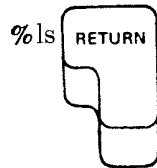
This is the end of Section VI. To continue, go to Part A of Section VII. To stop, go to Section IX for termination procedures.

VII. MORE EDITOR CONCEPTS

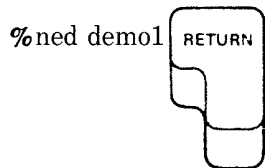
If you are continuing from Section VI, skip the following steps and begin with Part A of this section. Otherwise log in according to the directions in Section V and proceed with the steps below. If you decide to terminate the session, follow the directions in Section IX.



Copy the demonstration files into your directory.



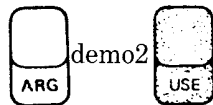
List files—demo1, demo2, and demo.alice should be in your directory.



Enter the Editor to view file demo1.

PART A. CALLING A FILE

After editing one file, a user may want to edit another file. The following edit command sequence brings up a new file.



This Editor sequence causes the Editor to put aside your current file, demo1, and to call demo2 for editing; demo1 then becomes the *alternate* file.



You can easily return to the alternate file demo1 by pressing the USE button; demo2 now becomes the alternate file.



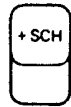
Using the USE function again brings you back to demo2. Only two files can be accessed concurrently in this manner. If you now were to use some other file, demo2 would become the alternate; demo1 would be neither the current file nor the alternate.

PART B. STRING SEARCHING

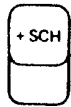
To assist editing, a search command is available to look for occurrences of a particular string.



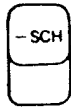
Here demo2 is searched for the first occurrence of the letters *specified*.



The +SCH function causes the Editor to search for the next occurrence of the last given string, which in this case was *specified*.



When there are no more occurrences of a string, the Editor issues the message: *Search key not found*.



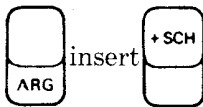
The -SCH function causes the Editor to search backward in the file from the current cursor position for *specified*.

PART C. ADDING AND DELETING LINES

This section demonstrates adding and deleting lines and partial lines. When adding new lines using the OPEN function, the lines are added above the line where the cursor is currently positioned and the cursor is positioned on the first of the new lines. The CLOSE function deletes the line where the cursor is currently positioned.



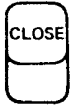
Go to the beginning of the file.



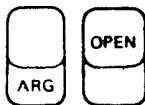
Locate the first occurrence of the word *insert*. Notice the cursor position.



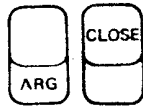
The OPEN function causes a blank line to be inserted immediately above the current cursor line.



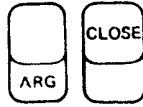
The CLOSE function deletes the current cursor line.



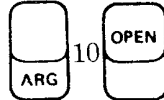
This sequence causes the text in the current line to the right of the current cursor position to be moved to the beginning of a newly inserted next line.



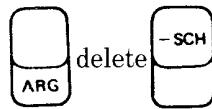
Following the above Editor command with this command will reverse the above operation. The line under the current cursor line will be moved to the current cursor line and positioned to the right of the cursor position.



Notice that existing text to the right of the cursor is deleted, then replaced by the next line.



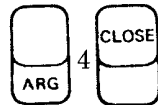
This Editor function sequence creates 10 new lines.



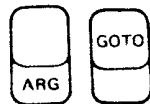
This Editor command causes a backward search in the file for the string *delete*.



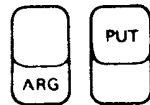
Move the cursor up 1 line.



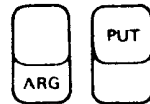
This Editor command deletes 4 lines starting with the current cursor position and automatically saves these lines in the CLOSE buffer. If 100 were used as an argument, lines not currently visible on the screen would also be deleted and put in the buffer. Only the information from the last CLOSE is immediately available using ARG PUT.



Go to the end of the file.



This Editor command places at the current cursor position the lines deleted by the most recent use of the CLOSE function.



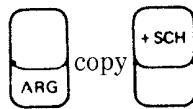
Another copy of the CLOSE buffer may be put in your file. This is a good way to make multiple copies of text.

PART D. PICK AND PUT

PICK and PUT functions copy lines, partial lines, or areas, and PUT them in a new location.



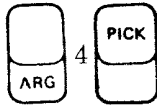
Go to the beginning of the file.



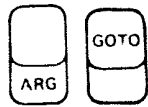
Locate the first occurrence of the string *copy*.



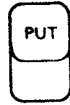
Move cursor up 1 line.



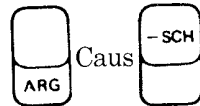
Take the current cursor line and the next three lines and place them in the PICK buffer. Using a larger argument allows lines not visible in the window to be PICKed and put into the PICK buffer.



Go to the end of the file.



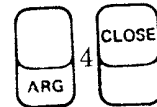
Put the contents of the PICK buffer at current cursor position.



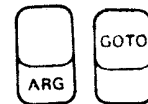
Search the file backward for occurrences of *Caus*; notice that PICK does NOT delete lines.



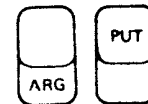
Move cursor up 1 line.



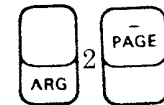
Delete the 4 lines describing PICK.



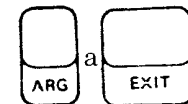
Go to the end of the file.



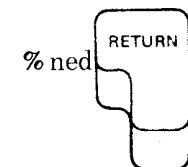
Put contents of CLOSE buffer here. (ARG PUT means to use the last CLOSE buffer, whereas PUT means to use the last PICK buffer.)



Back up 2 pages. Notice that the PICK definition was deleted.



Using *a* as an argument to the EXIT function causes an exit from the Editor without adding any of the modifications to file demo2 or any other files edited since last entering the Editor.



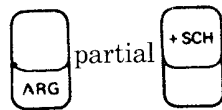
The % indicates that UNIX is ready to accept commands. Using *ned* without a file name causes the Editor to view the last file used, which is demo2.

PART E. CURSOR MOTION BUTTON ARGUMENTS

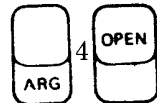
There are three types of arguments that may be used with Editor functions—character strings, numeric strings, and cursor motion buttons (CMB). This section illustrates some of the uses of CMB arguments. The number of times a single CMB should be pushed will be indicated in the explanation on the right. The button may be held down while the cursor moves the indicated number of positions or depressed the indicated number of times moving the cursor one position at a time. The examples given in this section should be considered as a guide and need not be followed precisely.



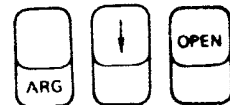
Go to the beginning of the file.



This is an example of a character string argument. A search is made for the first occurrence of the character string *partial*.



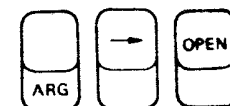
This is an example of a numeric string argument. Four new lines are inserted.



Push the down CMB 5 times. This is an example of a CMB argument. The cursor's movement defines the number of new lines to be added. Five new lines are added using this sequence.



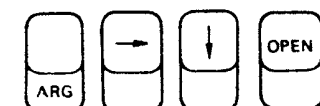
Push the down CMB 9 times. This sequence deletes the 9 lines defined by the cursor's movement. The space created by the two previous OPENs should now be deleted.



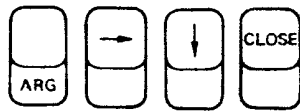
Push the right CMB 6 times. This sequence causes the text, to the right of the cursor, in the current cursor line, to be moved 6 spaces to the right.



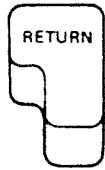
Push the right CMB 6 times. This Editor command deletes the 6 spaces defined by the cursor's movement in the current line.



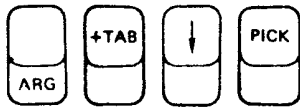
Push the right CMB 7 times. Push the down CMB 4 times. Using this series of cursor motion buttons will cause a new blank rectangular area defined by the cursor's movement to be inserted. The horizontal size of the rectangle is defined by the right CMB, and the vertical size is defined by the down CMB.



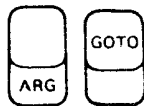
Push the right CMB 7 times. Push the down CMB 4 times. This sequence will delete the rectangular area defined by the cursor's movement. In this example, the space just inserted by the previous OPEN will be deleted.



Push the RETURN button twice.



Push the +TAB CMB 3 times. Push the down CMB 2 times. This series of buttons causes the area defined by the cursor motion buttons to be put in the PICK buffer.



Move the cursor to the end of the file.

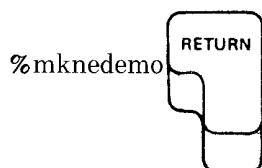


The PUT Editor function puts the contents of the PICK buffer at the position indicated by the current cursor position.

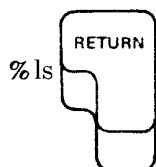
This is the end of Section VII. Section IX describes termination procedures. If you wish to continue, go on to Section VIII.

VIII. MULTIPLE WINDOWS

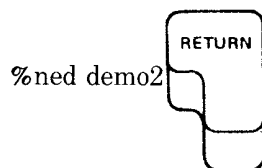
If you are continuing from Section VII, skip the following steps and begin with Part A; otherwise log in according to the directions in Section V and continue with the steps below.



Copy the demonstration files into your directory.



List files—demo1, demo2, and demo.alice should be in your directory.



Enter the Editor to work on demo2.

The New Editor can divide the screen into multiple windows with the WIN function. This allows the user to have more than one file, and/or different parts of the same file, available for editing.

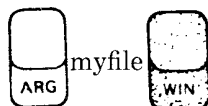
The steps in this section should be followed very precisely. If there are any problems, EXIT from the Editor and start over again after typing in *ned* to edit file demo2.



Go to the beginning of the file.



Move cursor to line 18. Check the message at the bottom of the screen for the correct line number. The cursor's position in column 1 indicates where the horizontal boundary for the new window is to be made.



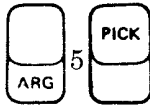
This function sequence forms a new window called *myfile*. A message will appear on the screen indicating that using the USE function will formalize myfile as a file.



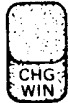
The screen is now divided in half. The top half of the screen contains file demo2. The bottom half of the screen contains the new empty file myfile. The cursor is located in myfile—the active file. (The active file is indicated by dominant window boundaries.)



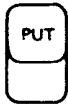
The CH WIN function causes the cursor to move to the next window. In this case, the cursor moves to the top window.



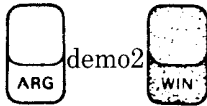
The user can communicate information between the two windows. PICK 5 lines of demo2 and put them in the PICK buffer.



Change windows; the cursor is now in myfile.



Put the contents of the PICK buffer into myfile. Move the cursor to approximately the middle of line 1 of the new window. This cursor position will define the left vertical boundary of the new window.



This Editor command forms a third window which contains demo2. There should now be three windows on the screen:

Window 1	demo2
Window 2 myfile	Window 3 demo2

kkkkk

Type in 5 k's.



Change to window 1. Notice that the changes made to window 3 were also made to window 1. (Even though demo2 is seen in two windows, there is only 1 copy.)



Remove the last window created—i.e., window 3.



Remove window 2. Now you should be left with window 1 and file demo2. The windows were stacked as they were created and are deleted in "last-in, first-out" order.

You have now finished Section VIII. Go to Section IX for termination procedures.

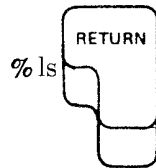
IX. TERMINATION

The UNIX command “mknedemo” copies the files demo1, demo2, and demo.alice into your own directory. These files may be deleted when you finish a session on the terminal or left in your directory, if you plan to continue the demonstration. It is a good idea to acquire the habit of deleting unnecessary files from your directory.

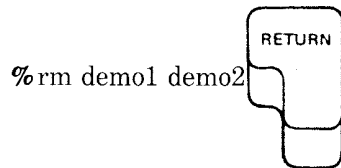
To delete your copies of the demonstration files before logging off, do the following:



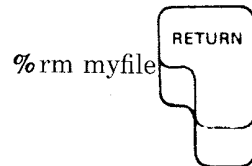
Exit from the Editor.*



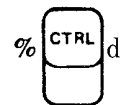
List your directory. Notice that there are backup files created by the Editor—i.e., files with names ending *.bak*.



Delete demo1 and demo2 from your directory.



Remove myfile from your directory and any other files no longer needed.



Log out from UNIX.

*If you decide not to apply the modifications made to your file during the session, use the following sequence for exiting from the Editor: ARGaEXIT.

X. AN EXAMPLE OF EDITING "ALICE"

In this section, a series of instructions is given to edit one of the demonstration files, demo.alice, so that it conforms exactly to the text of *Alice's Adventures in Wonderland*. In demo.alice there are many errors typically found in text. This section includes a copy of the original text, a copy of demo.alice, and a series of instructions to help edit demo.alice.

Original Text

This time Alice waited patiently until it chose to speak again. In a minute or two the Caterpillar took the hookah out of its mouth, and yawned once or twice, and shook itself. Then it got down off the mushroom, and crawled away into the grass, merely remarking as it went, "One side will make you grow taller, and the other side will make you grow shorter."

"One side of what? The other side of what?" thought Alice to herself.

"Of the mushroom," said the Caterpillar, just as if she had asked it aloud; and in another moment it was out of sight.

Alice remained looking thoughtfully at the mushroom for a minute, trying to make out which were the two sides of it; and, as it was perfectly round, she found this a very difficult question. However, at last she stretched her arms round it as far as they could go, and broke off a bit of the edge with each hand.

"And now which is which?" she said to herself, and nibbled a little of the right-hand bit to try the effect: the next moment she felt a violent blow underneath her chin; it had struck her foot!



She was a good deal frightened by this very sudden change, but she felt that there was no time to be lost, as she was shrinking rapidly: so she set to work at once to eat some of the other bit. Her chin was pressed so closely against her foot, that there was hardly room to open her mouth; but she did it at last, and managed to swallow a morsel of the left-hand bit.*

Copy of File demo.alice

This time alice waited patiently until it chose to speak again. In a minute or 2 the Cater pillar took the hook out of its mouth, and yawned once or twice, and shooook itself. Then it got down off the mushroom, and crawled away into the grass, meerly remarking as it went, "One side will make you grow taller, and the other side will make you grow fatter."

"One side of what? The other side of what?" thought Alice to herself. "Of the mushroom," said the Caterpillar, just as if she had asked it aloud; and in another moment it was out of sight.

Alice remained looking thoughtfully at the mushroom for a minute, trying to make out which were the two sides of it; and, as it was perfectly round, she found this a very difficult question. However, at last she stretched her arms round it as far as they could go, and broke off a bit of the edge with each hand.

She was a good deal frightened by this very sudden change, but she felt that there was no time to be lost, as she was shrinking rapidly: so she set to work at once to eat some of the other bit her chin was pressed so closely against her foot, that there was hardly room to open her mouth; but she did it at last, and managed to swallow a morsel of the left-hand bit.

Instructions for Editing File demo.alice

If the file demo.alice is not in your directory from the previous examples, execute the UNIX command *mknedemo* to copy this file into your directory for editing. Invoke the Editor to edit file demo.alice.

Follow these directions sequentially to edit the demonstration file. The line number refers to the line the cursor should be in before initiating the instructions that follow. At the end of these instructions, the demonstration file should look exactly like the original selection.

Line 1: Capital Letters. Move cursor under *a* in *alice* and type a capital A to form *Alice*.

Line 2: Adding Characters and Closing Spaces. Move cursor under 2, press INSERT MODE, press DEL CHAR, and then type in *two*.

Move cursor under space between *Cater* and *pillar*, press DEL CHAR to form *Caterpillar*.

Move the cursor under the space after *hook* and type *ah* to form *hookah*.

*Lewis Carroll, *Alice's Adventures in Wonderland and Through the Looking-Glass and What Alice Found There*, Henry Altemus, Philadelphia, Pennsylvania, 1895, pp. 69-70. Illustration from the original by John Tenniel.

Line 3: Deleting Extra Character(s). Move cursor under an *o* in *shook* and press DEL CHAR to form *shook*.

Line 5: Reversing Two Letters. Move cursor under second *e* in *meerly* and type *re*. Hit the DEL CHAR button twice leaving *merely* correctly spelled. (The same result could be obtained in the OVERTYPE MODE by typing *re* over the *er* in *meerly*.)

Line 6: Replacing a Word with a Longer Word. Move the cursor under the *f* in *fatter*. Press the DEL CHAR three times deleting *fat*. Type in *shor* replacing *fatter* with *shorter*.

Line 9: Making Two Paragraphs Out of One. Move cursor under the first “ and press ARG OPEN causing the text to the right of the cursor to move to a new next line.

Press RETURN and then hit the space bar eight times to form the new paragraph indentation.

Press OPEN to form a blank line between the two paragraphs.

Line 14: Adding a New Paragraph. Use the Editor command ARG 5 OPEN to add blank space for a new paragraph. Type in the following paragraph exactly as it appears here. Use the +TAB key for the paragraph indentation and finish each line using the RETURN key.

"And now which is which?" she said to herself, and nibbled a little of the right-hand bit to try the effect: the next moment she felt underneath her chin a violent blow; it struck her foot!

Line 14: Moving a Paragraph. Place cursor anywhere in line 14 and press ARG 5 CLOSE deleting this paragraph.

Move cursor to line 21 and press ARG PUT moving the deleted paragraph to its correct location.

Line 23: Reversing Two Phrases. Place cursor under the space before *underneath*. Press ARG, move the cursor until it is under the space after *chin*, and press CLOSE, deleting the phrase *underneath her chin*.

Move the cursor under the ; and press ARG PUT, moving the deleted phrase to its proper position.

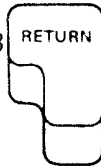
Line 27: Deleting a Line. Press the CLOSE button deleting this duplicate line.

Line 29: Adding Punctuation. Move the cursor under the space after *bit* and type a . to end the sentence. Press the INSERT MODE button, which switches to OVERTYPE MODE, move cursor under the *h* in *her*, and type *H* to form *Her*.

XI. RECOVERY

If there is a problem with the Editor, a message will appear on the screen telling you that the Editor just died. You can recover from a failure and restore your file using the following method:

%cp /tmp/retty.name demo3



Copy the Editor's temporary copy of your modifications to a file of your own—demo3 or any unused file. (Type in your log in name instead of *name*.)

%ned —demo3



Invoke the Editor to apply this file of your modifications to the backup copy of the file that you were editing. The — indicates to the Editor that demo3 is to be applied to the file that you were editing when the failure occurred. This procedure takes a little time, so wait patiently for a few moments before continuing.

R-2000-ARPA

