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

R-2176-ARPA
December 1977

The CRT Text Editor NED — Introduction and Reference Manual

Walter Bilofsky

A report prepared for

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

Rand
SANTA MONICA, CA. 90406

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

R-2176-ARPA
December 1977

The CRT Text Editor NED — Introduction and Reference Manual

Walter Bilofsky

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.

Published by The Rand Corporation

PREFACE

Members of Rand's Information Sciences Department are engaged in an ongoing research program on advanced intelligent terminals. The program is funded and coordinated by the Information Processing Techniques Office (IPTO) of the Defense Advanced Research Projects Agency (ARPA). An objective of the program is the improvement of the interface between man and computer.

This report documents a CRT text editor, called Ned, which is one of the software tools developed under this program. The report is intended to be an introduction to the Editor and a reference manual for experienced users. Portions of the report provide a description of Ned and summary of its features for a more general audience.

Ned was designed and implemented by the author, a member of the staff of Bolt Beranek and Newman, Inc., under subcontract to The Rand Corporation, whose Information Sciences Department conceived and sponsored the project.

SUMMARY

A critical element in the use of computers to support document preparation, software development, and word processing is the interface between man and machine for the entry and modification of textual data files. In recent years, the cathode ray tube (CRT) terminal has grown more popular as an interface as it has become more sophisticated and less expensive. In most applications, the CRT terminal is used only to simulate line-at-a-time devices such as teletypes. However, this device is also capable of displaying textual documents in page form, and can alter its display rapidly enough to allow interactive text editing.

Over the past few years Ned, a text editor utilizing the full capabilities of the CRT display, has been under development and in use at The Rand Corporation. It has been used primarily as a general text editor for system software development, and has also been of substantial benefit in the preparation of documents and reports by both technical and secretarial staff. The system has proven to be easy to learn and use.

The Ned editor runs on the PDP-11 series of computers under the UNIX operating system. It uses a CRT display to provide a two-dimensional window into a text file. The file may be altered by typing new material over the old. Lines, characters, and rectangular portions of text may be opened, deleted, and moved about, much as documents are edited with scissors and paste. Interactive text processing operations are provided, including paragraph fill, right-margin justification, and hyphenation. The set of operations may be expanded by user-provided or system-provided text-processing programs which are invoked interactively from within the Editor. The screen may be divided into several editing windows to simultaneously edit more than one file, or different portions of a file. The command language is based on function keys; thus, the terminal keyboard itself provides a list of editor functions.

This report contains a general description of Ned, a hands-on tutorial for beginning users, and the reference manual for Ned.

ACKNOWLEDGMENTS

Ned is based on concepts originally devised elsewhere by Edgar T. (Ned) Irons, currently at Yale University. The design was influenced by suggestions from many members of Rand's Information Sciences Department, and especially from Peter Weiner, who conceived the notion of implementing Ned at Rand, and who modified the program during its evolution. The error message chapter of this report was extended from earlier documentation by Suzanne Landa. The author is grateful to Paul Castleman of Bolt Beranek and Newman, Inc., for encouraging his participation in this effort.

CONTENTS

PART I. INTRODUCTION TO NED

Chapter 1

HOW TO USE THIS MANUAL

This manual is in fact two documents under one cover. The first, Part I, is designed primarily as an introduction to Ned, its history and description. Chapter 4 is a learner's guide to Ned, in tutorial form, suitable even for the user who is unfamiliar with computers and terminals. A reader with no previous exposure to computers or text editors may find it helpful to first read Sec. 4.1. This section might be skipped by a person who has had some computing experience. Part II is the reference manual for Ned, and its use requires familiarity with Ned in particular, and with computers in general. Hence, depending on his or her objectives and level of experience, the reader may want to begin with different chapters of the manual.

The beginning user should gain access to a PDP-11 terminal and follow the tutorial in Chap. 4, trying things out on the terminal.

Ned users with almost any degree of experience may find this document more valuable as a reference manual, and may make use mainly of the function reference chart (App. A), the Index of Capabilities (App. C), and the Reference Manual (Part II).

Persons seeking an understanding of Ned who do not have access to a terminal capable of running Ned may find it most useful to read the brief description of Ned in Chap. 2, the Index of Capabilities (App. C), and the Reference Manual (Part II).

The reader should be aware of one typographic convention used throughout this manual. An upper case legend (e.g., +TAB) represents the function button that is labeled by that legend on the terminal keyboard. Thus, "Type OPEN" means "Press the key marked OPEN." Some of the keys on the keyboard contain both printing characters and a function legend. The CTRL key is used as a shift key for typing the functions on these keys. (The only exception to this is the EXIT key, which does not require a CTRL shift). Sometimes a function key will be described in the form, "CTRL-X". This function is typed by holding down the CTRL key while pressing the X key.

While some terminals may not have all the functions marked on the keyboard, the keys will still perform the functions. The chart in App. B may be used to locate the function keys.

Chapter 2

BRIEF DESCRIPTION OF NED

Ned (the New Editor) is a text editor which allows the user to view and edit text files in an easy and natural way. This chapter describes Ned's capabilities, assuming the reader to have a small amount of experience as a computer user.

Ned runs on a Digital Equipment Corporation PDP-11 computer under the UNIX operating system, using the Ann Arbor 4080D CRT terminal. Outside of The Rand Corporation, Ned has been modified by users to run on certain other models of CRT terminals.

Ned presents the user with a picture of the text file being edited, just as it would appear when printed out on paper. The terminal screen contains a window which can be moved about the file to show any part of it. Changes are made directly on the screen and are shown as they are performed. Operations such as typing diagrams are easy, since the user is free to enter characters at any position on the screen, rather than having to type text line by line. The command language consists of function buttons which are marked with the operations they perform. The net effect is that Ned places the user in direct contact with the document, unlike line-oriented editors, which interpose two conceptual barriers—a command language and editing a line at a time—between the user and the document. This makes Ned easier to learn and use than most other text editors.

Changes can be made to a file by typing directly on the screen. A character is changed by typing the new character over it. More complicated operations are possible, such as inserting, deleting, or moving characters or lines. It is possible to insert, delete, or move rectangular pieces of the document, using the computer terminal as one would use scissors and paste to edit a paper document.

Advanced features include interactive text formatting, such as automatic fill, right-margin justification, and hyphenation of text paragraphs. An extensibility feature allows the expansion of Ned capabilities through the writing of UNIX filter programs to process text. These programs can then be used interactively to process paragraphs of text during an editing session.

The screen may be divided into several editing "windows" to simultaneously examine or edit more than one file, or different portions of a file.

Ned provides several levels of backup to guard against inadvertent destruction of files. All lines deleted during an editing session are accessible until the end of that session. The second most recent version of an edited file is preserved under a backup name. In the event of a computer hardware or system malfunction, the editing session can be reproduced, generally without loss of a single keystroke.

Chapter 3

HISTORICAL DEVELOPMENT OF NED

This chapter gives a brief history of Ned and its predecessor CRT editors. It is intended to record Ned's development, and not as an exhaustive history of CRT text editors.

The IDA-CRD (Institute for Defense Analyses—Communications Research Division) text editor was invented by Edgar T. (Ned) Irons in 1967 to provide text editing support for software development of the IDA-CRD operating system for the CDC 6600 computer. The IDA editor is described in E. T. Irons and F. M. Djorup, "A CRT Editing System," *Communications of the ACM,* January 1972. The IDA editor contributed the idea of function keys as an editing command language, and of using the screen as a two-dimensional representation of the file. For the individuals eventually responsible for Ned, it was the first, and a convincing, demonstration of the utility of two-dimensional editors.

The Yale editor "E" was implemented by Peter Weiner in 1970 on a DEC (Digital Equipment Corporation) PDP-10 computer. It was the successor to the IDA editor, and proved the practicability of CRT editing over lines of modest bandwith (2400 baud) on standard terminals. The Yale editor is described in *The Yale Editor "E"—A CRT Based Editing System,* P. Weiner, I. Singh, D. J. Mostow, and E. T. Irons, Yale Computer Science Research Report 19, April 1973.

The Rand editor, "re", was conceived in 1974 by Peter Weiner, based on the Yale editor. It was initially designed and implemented by the author of this report, and modified by both individuals as it evolved. The Rand editor is now known as Ned, which stands for "new editor" and is also a tribute to the inventor of the IDA editor. Features appearing in Ned but not in its predecessors include multiple editing windows and interactive text processing using user-provided as well as system-provided functional modules.

Ned has been in use at The Rand Corporation and other UNIX installations since 1974. It is used by technical, administrative, and secretarial personnel in the daily performance of their jobs.

Chapter 4

NED TUTORIAL

The best way to learn about Ned is to sit down with a terminal and try Ned out. Any explanation will make the editor sound far more complicated than it really is. On the other hand, just using Ned is really quite easy. Learning to use it is like learning to use a typewriter. Rather than reading a manual on how to type, one gets a better sense of the machine by sitting down, inserting a piece of paper, and trying things out.

Accordingly, this introduction for the new Ned user presents a few simple directions to help get started—instructions for turning Ned on and inserting a sheet of blank paper, as it were—and a series of things to try out.

The exercises presented below are constructed to expose the user to a basic set of Ned's features by leading him/her through a sequence of operations demonstrating those features. It is hoped that the user will not feel limited to these steps, but will feel free to try different functions and combinations as well.

4.1. Introduction to Computers and Text Editors

The purpose of this section is to provide some background concepts and definitions to potential users of Ned who are not at all familiar with computers or text editors.

The computer can be viewed as an automatic file cabinet. It stores many collections of information, called *files*. A file is analogous to a file folder in a filing cabinet. Just as a file folder has a name on its tab, a computer file has a *file name*.

A computer file can contain a program, or numeric data, or textual data. A file which contains textual data is called a *text file*. A text file may be used to store a typed document, such as a memorandum or report.

Typed documents on paper may be edited in several ways: with correction fluid, with scissors and paste, or by retyping. Text files are edited using a tool called a *text editor*. This is a program which lets the user manipulate the text file.

Ned is a *two-dimensional CRT text editor*. CRT (Cathode Ray Tube) is the technical name for a television picture tube. Ned uses a *CRT terminal*—a computer terminal with a typewriter keyboard and a television-type display. *Two-dimensional* means that text is displayed a screenful at a time (as opposed to one dimensional —a line at a time).

4.2. Accessing the PDP-11

Ned is run from a terminal that has been connected to the PDP-11 system and logged on to UNIX. The procedure for doing this at Rand is explained here; this procedure will vary at other installations.

The terminal must be turned on (power switch on the back). After warming up,

it will display a green underscore character or *cursor*. The message "User:" should also be displayed. If it is not, press the RETURN key. If the message still does not display, the terminal is not connected to the PDP-11; consult an experienced user for assistance. If the character "%" displays when RETURN is pressed, a user is already logged on to the terminal and the procedure in the next paragraph may be skipped.

When the message "User:" appears, type your user account name. If the user name is acceptable, the message "Password:" will be displayed. Type your account password. What you type will not appear on the screen, but is nevertheless entered into the computer. If the password is valid, the terminal may display a short message of current interest to system users, indicate the last time you logged on to UNIX, and then indicate its readiness to accept a command by displaying the character "%".

You may want to learn more concerning the use of the PDP-11 UNIX system than is within the scope of this manual. The first place to look is in "UNIX for Beginners," by Brian Kernighan (included in *Documents for Use with the UNIX Time-Sharing System,* available from the Rand Documentation Center).

4.3. Accessing Ned

This section explains how to start using Ned from a terminal which has been connected to the PDP-11 as described in the previous section. If the terminal does not have all the keys marked with their editing functions, refer to the keyboard diagram (App. B) to identify which keys perform each function.

The first thing necessary is a "piece of scratch paper" to type on. In other words, a scratch file is needed. So we need to find a file name that is not currently in use as the name of an existing file.

See if "fname" may be used as a scratch file name. This can be done by executing a UNIX command which tells whether there is already a file named fname. If there is not, fname may be used as a scratch file name. If fname exists, another name must be used.

Type the UNIX command "ls fname" followed by the RETURN key. (UNIX commands are typed using only lowercase letters; ls stands for "list".) The system responds either with "fname", meaning there is a file named fname, or with "fname not found", meaning there is no file named fname. We will assume the name "fname" was available, and proceed. If there is a file named fname, try "ls gname" or "ls newfile" or any other name until some name is discovered which is "not found" and therefore is free for use as a scratch file name. Wherever "fname" appears in the exercises below, use your scratch file name instead.

Cursor. Notice that there is an underscore character on the screen to the right of the % character. This character is called the *cursor*. It indicates the position on the screen at which the next character typed will appear.

To begin a Ned session, type the UNIX command "ned fname" followed by the RETURN key. (Use lower case: "ned", not "Ned".) Ned displays a message at the bottom of the screen saying "Hit USE (CTRL-B) to make: fname." Type the USE key by holding down CTRL (like a shift key) and typing b. This creates file "fname". Now Ned is ready for experimentation. What you will see on the screen is the Ned editing "window."

4.4. First Exercise: Exiting from Ned

EXIT. Press the EXIT key. (It is marked DEL on some keyboards.) Ned exits, and UNIX types % again to indicate it is ready for a UNIX command. Start Ned up again, by typing "ned fname" and RETURN. Since the file named fname has already been created, this time Ned will not ask for it to be made.

Every time Ned is exited, the file that was being edited is saved. The file remains in existence indefinitely, or until it is removed.

4.5. Second Exercise: Cursor Positioning Keys

The cursor is very important in the Editor. It can be moved to any position on the piece of file being viewed in order to type text at that position.

Positioning the Cursor. The keyboard contains eight *cursor positioning keys.* These are the four keys marked with arrows pointing up, down, left, and right, and the HOME, −TAB, +TAB, and RETURN keys. Experiment with them and observe how they move the cursor. Caution: Try not to press the RETURN key when the cursor is positioned on the bottom line. This causes the window to move down the file, and may be confusing. An explanation is given in Sec. 4.7.

The lower right-hand corner of the screen displays the number of the line in the file on which the cursor is positioned. Watch the number as you move the cursor up and down.

Hold a positioning key down for a few seconds and notice how it repeats. Every key on the terminal repeats if it is held down for more than a second.

4.6. Third Exercise: Typing

Typing Over Something. Characters are inserted into a file by typing them onto the screen. Type something on the screen using the keys that contain letters, numbers, and punctuation. The cursor positioning keys can be used to place the cursor at any position on the screen, in order to type there.

BS. Position the cursor under some typed characters and type over them. Characters may be deleted by typing over them with the space bar. Type something and then use the BS (backspace) key. Note that the BS key is different from the "move left" (left arrow) positioning key.

DELCHAR. Position the cursor in the middle of some typed words. Press the DELCHAR key and note what happens.

INSERTMODE. Press the INSERTMODE key. Note the message on the bottom line which indicates that Ned is now in *insert mode.* Position the cursor in the middle of some typed words and type something. Notice what happens. Try the BS key. Insert mode only affects what happens when a character or BS is typed; all the positioning and other keys work the same as before.

To exit from insert mode, press the INSERTMODE key again. Notice that the message at the bottom of the screen has disappeared.

4.7. Fourth Exercise: Moving the Window

The file being edited can be thought of as a long scroll. The area on the screen enclosed by the borders is called the *editing window*. It shows 37 lines of the file at any time. This window can be moved up or down the file in order to edit different parts of it. Type a few characters on each line of the screen before proceeding further, so that the effects of moving the window are visible. For instance, you could type "Line 1" on the first line, "Line 2" on the second line, etc.

+LINE and −LINE. Press the +LINE key. This moves the window down (i.e., the file up in the window) about a quarter of the screen. The −LINE key moves in the other direction (but not past the first line of the file). The lower right-hand corner of the screen shows on which line number of the file the cursor is positioned. This helps to locate where the window is pointing once it has been moved around.

Position the window so that the first line of the file is at the top of the screen. This can be done by pressing −LINE until the window will not move any more. Then move the window down using +LINE until there are some blank lines at the bottom. Type something on these lines.

+LINE and −LINE can be pressed again before Ned has finished rewriting the screen from the first push. This can be handy if you want to move the window repeatedly while watching to see if a particular spot has been reached. Be careful not to hold down the +LINE or −LINE function key, as the key will repeat if held, causing more window motion than might be intended.

RETURN on Bottom Line. Position the cursor on the bottom line of the window and press RETURN. The window will move down just as if +LINE had been pressed. This enables continuous text entry at the end of a file without the need to keep moving the window down to bring in new blank lines.

+PAGE and −PAGE. Position the window at the first line of the file again. Press +PAGE. Notice that this moves the window down 37 lines, or one page. (The term *page* has nothing to do with paper pages: it merely means one window's worth of lines.) Press +PAGE again, moving the window down another page, and repeat until a completely blank page appears. Notice the ";" characters in the left margin. They indicate that the window is positioned past the last typed line of the file. Press −PAGE repeatedly to move the window back to the first line of the file. Notice the characters in the left margin.

Experiment with +LINE, −LINE, +PAGE, and −PAGE to position the window at various places in your file.

ARG. Press the ARG key (marked BRK on some terminals). Notice that the cursor moves to the bottom of the screen. Type a few characters. Try the BS key. The ARG key is used to enter *arguments*, which are values which modify the various functions. Now press ARG again. Notice that whatever was typed at the bottom of the screen has disappeared. This illustrates that pressing ARG while typing an argument serves to wipe out the partially typed argument.

Press the ARG key, type the number 2, and press +LINE. The window moves 2 lines down the file. The ARG key and numeric argument "2" served to modify the very next function key, +LINE, by telling it how many lines to move. Try different numeric arguments for the +LINE, −LINE, +PAGE, and −PAGE functions.

4.8. Fifth Exercise: Moving Lines Around

OPEN. Position the cursor on a line containing typing. Press the OPEN function key. A blank line opens up. Press OPEN several times in succession. It is not necessary to wait for Ned to finish writing the contents of the screen before the next function key is pressed.

CLOSE. Position the cursor on a blank line and type CLOSE. This deletes the blank line. Now position the cursor on a line of typing that is not needed, and press CLOSE. This deletes the line.

PICK and PUT. Position the cursor on a line with typing, and press PICK. Nothing visible happens, but a copy of that line has been picked up and stored inside the editor. Position the cursor on another line and press PUT. A copy of the picked-up line is inserted there. Type PUT again. Another copy of the line appears. Multiple PUTs may be used to make any number of copies of a line.

Use CLOSE to delete a line of typing. Now position the cursor on another line and press ARG followed by PUT. A copy of the deleted line is inserted there.

PICK and PUT operate by using two "buffers", or internal bins for temporarily storing pieces of text. These are called the *pick buffer* and the *close buffer*. PICK picks up a copy of a line and places it in the pick buffer. PUT puts the contents of the pick buffer back into the file. CLOSE deletes a line, and also places it in the close buffer. ARG PUT puts the contents of the close buffer back into the file.

Remember that the ARG key was used to specify a number of lines for +LINE and −LINE, or a number of pages for +PAGE and −PAGE. In the same way, pressing ARG, a number, and OPEN will open up that number of lines. Similarly, ARG, a number, and CLOSE or PICK will close or pick that number of lines. Try opening, closing, and picking more than one line at once by using ARG and a number together with the function key.

4.9. Summary

With the features of Ned that have been demonstrated in this chapter, the beginning user can create and edit files on the UNIX system. For handy reference, these features are summarized in the first portion of the chart in App. A.

Ned is also capable of performing many editing operations not covered in this chapter. The Index of Capabilities (App. C) provides a summary of these operations and page references to the sections of the Reference Manual (Part II) which explain these operations in more detail. The user should browse through App. C in order to become acquainted with the capabilities of Ned.

Another tutorial introduction to Ned, which covers features not touched on above, is *A Guide to Ned: A New On-Line Computer Editor* by Jeanne Kelly, The Rand Corporation, R-2000-ARPA, July 1977. That report includes a model Ned session.

PART II. NED REFERENCE MANUAL

Chapter 5

REFERENCE MANUAL: INTRODUCTION

This reference manual is intended to provide answers to specific questions about Ned. The user will probably wish to start looking for information in one of three places: the Summary of Ned Function Keys (App. A), the Index of Capabilities (App. C), or the index. The Index of Capabilities lists the editing operations which Ned can be used to perform. It is intended to make it possible for the user to answer questions of the form, "How do I do so-and-so in Ned?"

The remaining chapters of the reference manual appear in a moderately logical order, but are not intended to be read in any particular sequence.

Chapter 6

RUNNING NED: INITIAL ARGUMENTS

Ned can be initialized in several different ways, depending on the arguments given to the "ned" UNIX command.

ned filename: Ned is initialized looking at the first page of the indicated file. If the file does not exist, the message "Hit USE (CTRL-B) to make: filename" appears (see USE, Sec. 11.9).

ned: Ned is initialized to the file and position that were displayed the last time the user exited from Ned. If the user has not previously run Ned on that day, Ned will initialize looking at the system file /re.std, which contains a message indicating that there is no default file specified. If multiple windows were in use, only the file in the last active window is displayed.

ned –ttyfile: ttyfile must be the name of a Ned keystroke file (Sec. 13.3). Ned duplicates the editing session from the keystroke file.

ned +filename: Ned is initialized pointing at the indicated file, and the last current file and position from the previous Ned session are now used as the alternate file (Sec. 11.9).

ned !: Ned is initialized precisely as it appeared at the end of the last Ned session. All windows, files and alternates are restored.

ned !filename: Ned is initialized as for "ned !", but the named file is shown in the last active window, and the file that was in that window becomes the alternate file.

In any of the above examples, the file specifier "filename" can actually consist of up to four arguments, as follows:

filename: specifies a file.

filename line: "line" is an integer specifying the line number at which the top of the window is to be positioned.

filename line col: "line" specifies the line number, and "col" the column number, at which the upper left-hand corner of the window is to be positioned.

filename line col srchkey: "line" and "col" specify the line and column number at which the upper left-hand corner of the window is to be positioned. Then a +SCH from that point is executed to find the next occurrence of the search key "srchkey". To specify a search from the beginning of the file, "filename 0 0 srchkey" may be used.

Chapter 7

MARGIN CHARACTERS

Ned uses a number of different characters to draw the borders around its editing windows.

| is used for left and right margins when no other character applies.

_ is used for top and bottom margins.

; appears in the left-hand margin to indicate that the line on which it appears is past the end of the file. Text may be typed on such a line, and the file will be extended to that line, although Ned does not replace the ";" on any line by "|" until that line has to be redisplayed because it has been changed or moved.

< appears in the left-hand margin to indicate that the window has been moved right, and there is text (perhaps all blanks) to the left of the left-hand margin of the window. ";" takes precedence over "<".

> appears in the right-hand margin to indicate that there is text (not all blanks) on that line beyond the right-hand margin of the window.

. appears in the margins of a window which is not active, when there is more than one window on the screen. It does not appear in margins which are shared with an active window.

■ A bullet (bright rectangle) is displayed in each margin to indicate the column and line containing the cursor. (This is suppressed for terminals running at slower than 9600 baud, since displaying the bullets would slow down the display.)

Chapter 8

CURSOR POSITIONING KEYS

There are eight cursor positioning keys. Four are marked with arrows pointing left, right, up, and down. These keys move the cursor one space in the indicated direction. When the cursor reaches the boundary of a window, it "wraps around," moving to the opposite margin. Holding the keys down for a second causes them to start repeating their function fifteen times per second.

The HOME key moves the cursor to the upper left-hand corner of the window.

The +TAB key moves the cursor to the next tab stop to the right of the current cursor position. Tab stops are initially set at columns 9, 17, 25, 33, 41, 49, 57, 65, and 73. Tab stops may be cleared or set in any column by use of the S/RTAB function (Sec. 11.13).

The −TAB key moves the cursor to the next tab stop to the left of the current cursor position, except that pressing −TAB when the cursor is in column 1 will move the cursor to the last column of the window.

The RETURN key is usually a cursor positioning character, moving the cursor to the first column of the line below the current one. However, if RETURN is pressed when the cursor is on the last line of the window, a +LINE function will be performed before the cursor is moved. This enables continuous text entry at the end of a file without the need to keep moving the window down to bring in new blank lines.

Cursor positioning keys may be given positive numeric arguments: e.g., ARG 6 RETURN. The effect is to repeat the key the indicated number of times. This might be useful for moving the cursor many lines; the tab keys are usually sufficient for long horizontal moves.

Chapter 9

OTHER SPECIAL KEYS

This chapter describes four keys which fall outside the categories of normal typewriter key, cursor positioning key, and function key. They are INSERTMODE, BS, DELCHAR, and QUOT.

9.1. Insert Mode: INSERTMODE

INSERTMODE inverts the mode of the Editor. If the Editor is in normal mode, INSERTMODE puts it in insert mode. If the Editor is in insert mode, INSERTMODE returns it to normal (overtype) mode.

Insert mode affects the action of all printing characters (including space). In normal mode, a typed character is inserted at the cursor position, replacing any character that might have been there previously. In insert mode, a typed character is inserted at the cursor position, and all characters that were to the right of that position are moved over one position to the right. No characters are deleted. If there was a character in the last column, it moves out beyond the right-hand margin. It is no longer visible, but the margin character changes from | to > to indicate that there is text beyond the margin.

Insert mode also affects the action of the BS key (see next section).

9.2. Backspace: BS

BS (backspace) is similar to the space bar, except that it moves the cursor left instead of right. In normal mode, it moves the cursor left one position, and inserts a blank in that position. In insert mode, it moves the cursor left one position, deleting the character at that position and moving all characters to the right of the deleted character one position to the left.

Characters deleted by this key cannot be recovered automatically, and must be retyped to be recovered.

9.3. Deleting Characters: DELCHAR

DELCHAR deletes the character at the current cursor position, and moves all characters to the right of the deleted character one position to the left. The cursor does not move. Characters deleted by this key cannot be recovered automatically, and must be retyped to be recovered.

9.4. Inserting Control Characters: QUOT

Control characters are non-printing characters, some of which perform format-

ting functions in text files. The QUOT key allows the explicit insertion of control characters into text. This function is typed by holding the CTRL key down while striking the "\" key. This causes a bullet to be inserted on the screen. The next character typed should be another control character. It will appear as a printing character immediately following the bullet. The bullet and printing character act as two separate characters during the editing session, but when the file is saved they are written into the file as a single control character.

When a file that contains control characters is edited, these characters are displayed as a bullet followed by a printing character, and are written out again as the single control character.

The main use of this feature is to insert formatting control characters in a file. In particular, the character control-L (typed by QUOT L) will cause a form eject at that point when the file is printed on a line printer or other output device. Other formatting characters that can be used with some printers and terminals are control-H (backspace) and control-K (vertical tab).

Chapter 10

COMMAND LANGUAGE

Users who are familiar with other editors know that every editor is told which functions to perform by means of commands.

To tell one editor to delete a line, the user might type "DELETE". In the command language of another editor, the delete command might be given by simply typing "d".

Ned has an abbreviated command language, consisting of the *cursor position, cursor motions, arguments* and *function keys*. The terminal screen displays a *cursor*, which is an underscore indicating the position where the next character that is typed will appear. Eight cursor motion keys allow the user to move the cursor about the screen. Many editing functions are performed at the spot, or on the line, indicated by the cursor. Thus the first element of the command language is the cursor position.

Commands are performed by function keys. For example, to delete a line in Ned, instead of typing "DELETE" or "d", the user positions the cursor anywhere on the line to be deleted and presses the function key marked CLOSE.

10.1. Arguments to Commands: ARG

Many commands are made more flexible by use of arguments. For example, to delete (close up) five lines, the CLOSE function key is used, with an argument of 5. This is done by pressing the ARG key (which moves the cursor down to the bottom of the screen so the argument is not typed onto the file), typing the argument "5", and then pressing CLOSE. This can be written as ARG 5 CLOSE.

In general, a function is invoked with an argument by typing the ARG key, the argument, and the function key. While the argument is being entered, the cursor is positioned on the last line of the screen. A bullet temporarily holds the place on the screen that corresponds to the cursor position.

Some functions can take only arguments which are numeric. Others may take as argument any string not containing a positioning key (see Chap. 8). The table in App. A indicates the type of argument required by each function and the way in which the argument is interpreted. If an improper argument is given, the Editor will sound the bell (on terminals so equipped) and give an error message at the bottom of the screen.

While typing an argument, the BS key may be used to correct errors in typing. When all characters in an argument have been erased by BS, any further BS keys are ignored.

If a positioning key is typed in an argument, three different interpretations are possible. If only numeric characters have been typed in the argument, they are taken as the argument to the positioning key (Chap. 8). If non-numeric characters have been typed, an error message will be given. If a positioning key is the first key in an argument, the argument is taken to be a cursor-defined argument (see next

section). Thus, only BS, and not positioning keys, may be used to edit a partially typed argument.

To erase a partially typed argument without performing any function, type the ARG key again.

10.2. Cursor-Defined Arguments

Cursor motions can also be used as arguments, to indicate directly on the screen which lines, characters, or rectangular areas are to be manipulated. This amounts to indicating a set of lines or a rectangular area on the screen by saying "From here ... to here," where each "here" is indicated by a positioning of the cursor.

Indicating Lines by Cursor-Defined Arguments. A cursor-defined argument can be used to designate a group of lines to be operated on by OPEN, CLOSE, or PICK.

For example, deleting (closing) several lines by cursor positioning takes the following four steps: (1) Position the cursor somewhere on the first line to be deleted. This is the "From here." (2) Press ARG. (3) Position the cursor at the spot directly below on the last line to be deleted. Do this by pressing repeatedly on the key with the arrow pointing down on it. This defines the "to here." (4) Press CLOSE. This will delete the lines "From here ... to here."

Opening or picking several lines may be done in exactly the same way, using the OPEN or PICK function key.

The lines defined by vertical cursor motion are the lines from the initial to the final position, inclusive.

Indicating Areas by Cursor-Defined Arguments. A cursor-defined argument can be used to designate a rectangular area of text to be operated on by OPEN, CLOSE, or PICK.

In the example in the previous section, if the "to here" is not in the same column as the "from here," the argument defined by the cursor motions is not a series of lines, but rather a rectangular area on the screen. Such an area can be closed or picked. If it is opened or put, the line(s) in which the text is inserted are opened up horizontally to make room for the new material, and the remainder of each line is moved over to the right.

The area defined by vertical and horizontal cursor motions consists of the lines from the initial to the final position, including both the initial and final line, and the columns from and including the leftmost position, up to but *not* including the rightmost position.

It is not necessary for the "from here" to be above the "to here," or to its left. The only requirement is that a sequence of lines be defined by indicating the first and last lines in either order, or that a rectangle be defined by indicating two opposite corners in any order.

The only functions which take cursor-defined arguments to specify groups of characters, lines, or rectangular areas of text, are CLOSE, OPEN, and PICK.

Chapter 11

THE FUNCTION KEYS

This chapter explains in detail the effect of each function key in Ned, with the exception of the EXEC key, which is treated in Chap. 12. The effect of each function is summarized in the chart of App. A.

11.1. Vertical Window Motion: PAGE, LINE, and RETURN

In Ned the term *page* refers to the amount of text that fits in a window, usually 37 lines.

Pressing the function key marked +PAGE moves the window down the file a number of lines equal to one page. ARG n +PAGE, where n is a numeric argument, moves the window down the file n pages, i.e., n times the number of lines in the window. The cursor is positioned about one-fourth of the way down from the top of the window, in the leftmost column. If a zero argument is given, neither the window nor the cursor are moved, and if a negative argument is given the window is moved up the file the indicated number of lines, as for −PAGE.

−PAGE moves the window up the file a number of lines equal to one page. ARG n −PAGE, where n is a numeric argument, moves the window up the file n pages. The cursor is positioned about one-fourth of the way down from the top of the window. An attempt to move the window above the first line of the file will move it to the first line of the file. If a zero argument is given, neither the window nor the cursor are moved, and if a negative argument is given, the window is moved down the file the indicated number of lines, as for +PAGE.

+LINE moves the window down the file a number of lines equal to one-fourth of the number of the lines in the window, rounded up to the next integer. ARG +LINE moves the window so that the line containing the cursor is on the top line of the window. ARG n +LINE, where n is a numeric argument, moves the window down the file n lines. If the line containing the cursor is still contained in the window after the move, the cursor moves with the line, remaining in the same position relative to the text. If the line containing the cursor is no longer in the window, the cursor is positioned about one-fourth of the way down from the top of the window, in the leftmost column.

−LINE moves the window up the file a number of lines equal to one-fourth of the number of the lines in the window, rounded up to the next integer. ARG −LINE moves the window so that the line containing the cursor is on the bottom line of the window. ARG n −LINE, where n is a numeric argument, moves the window up the file n lines. An attempt to move the window above the first line of the file will move the window to the first line of the file. If the line containing the cursor is still contained in the window after the move, the cursor moves with the line, remaining in the same position relative to the text. If the line containing the cursor is no longer in the window, the cursor is positioned about one-fourth of the way down from the top of the window, in the leftmost column.

ARG +PAGE and ARG −PAGE, and cursor-defined arguments with any of the PAGE and LINE keys, will produce an error message.

If +PAGE, −PAGE, +LINE, and −LINE are used to move the window farther than 32,767 lines (about 885 pages) or past line 32,767 in the file, the editor will fail.

RETURN, when typed on the last line of a window, has the same effect as the sequence +LINE RETURN. This makes it possible to enter many lines of text at high speed without explicitly repositioning the window. When the cursor is on any line but the last, RETURN is only a cursor positioning key.

11.2. Opening Lines and Areas: OPEN

Pressing the function key marked OPEN inserts a blank line at the current cursor line, and moves all subsequent lines down.

ARG n OPEN, where n is a positive integer, inserts n blank lines at the current cursor line, and moves all subsequent lines down.

ARG OPEN splits the current line by moving the characters at and to the right of the cursor position onto a new next line, and moves all subsequent lines down one line. This action may be reversed by typing ARG CLOSE before repositioning the cursor.

ARG *motion* OPEN, where the argument is cursor defined, is used to open either a number of lines, or a rectangular area, as indicated by the cursor-defined argument (see Sec. 10.2). If lines are indicated, ARG *motion* OPEN opens that number of lines. If an area is indicated, blank spaces are inserted in the columns of that area in each line, and the remainder of the line is moved to the right to make room for the blanks. No characters are deleted, but characters may be pushed out of the window to the right. If this occurs, the right margin character changes from "|" to ">" to indicate that there is text beyond the right margin.

In all variations of the OPEN function, the cursor remains at the position it occupied before the function was initiated. When the ARG key is used, the cursor returns to its position before the ARG was typed.

11.3. Deleting Lines and Areas: CLOSE

The function key marked CLOSE deletes the line on which the cursor is positioned and places the contents of the line in the close buffer. The previous contents of the close buffer are lost. (See PICK, Sec. 11.4, for a discussion of buffers.)

ARG CLOSE deletes the part of the current line beginning with the cursor position and extending to the end of the line. The deleted portion is replaced by the contents of the next line; i.e., the remaining portion of the current line is merged with the next line. All lines below are moved up one line to fill the gap caused by merging the two lines. The deleted line fragment is not saved in the close buffer; the contents of the close buffer are not altered.

ARG n CLOSE, where n is a positive numeric argument, causes the n lines beginning with the current cursor line to be deleted. The deleted lines are placed in the close buffer, and the previous contents of the close buffer are lost. If the specified number of lines extends beyond the end of the file, a sufficient number of blank lines are supplied in order to provide a total of n lines to the close buffer.

ARG *motion* CLOSE, where the argument is cursor defined (see Sec. 10.2), causes the text in the indicated lines or area to be deleted. If lines are indicated, the effect is as if ARG n CLOSE had been typed, with n equal to the number of lines indicated. Areas are deleted by removing the indicated portion of each line, and moving left the characters to the right of the deleted portion to the left to fill the gap. The deleted lines or area are placed in the close buffer, and the previous contents of the close buffer are lost.

In all variations of the CLOSE function, the cursor remains at the position it occupied before the function was initiated. When the ARG key is used, the cursor returns to its position before the ARG was typed.

The text deleted by most CLOSE commands can be replaced in its original position by typing the sequence ARG PUT as the first thing typed after the CLOSE key (see PUT, Sec. 11.5). However, this will not work for ARG CLOSE which, as noted above, does not save the deleted text in the CLOSE buffer. Further, ARG PUT will not restore the former contents of the close buffer. However, anything placed into the close buffer is also placed in the # file (see Sec. 13.2), and thus can be recovered at any time during the editing session.

11.4. Replicating Lines or Areas: PICK

PICK is used with PUT to pick up a copy of the text in a group of lines or a rectangular area, without disturbing the text, and then to put one or more copies of the text down at any point in any file being edited. PICK picks up a copy of the text and places it in an internal storage bin called the *pick buffer*. PUT copies the contents of the pick buffer back into a window. (CLOSE operates similarly to PICK, but places its text into a different buffer, the *close buffer*, and deletes the text from the file at the same time.)

A buffer may contain lines of text, or an area of text. If the contents of the buffer were placed there by a cursor-defined argument which indicated a rectangular area of text, then the buffer contains a rectangle of text. Otherwise, it contains lines.

Pressing PICK places a copy of the current cursor line into the pick buffer.

ARG n PICK, where n is a positive integer, places the n lines beginning with the current cursor line into the pick buffer. If the specified number of lines extends beyond the end of the file, a sufficient number of blank lines are supplied in order to provide a total of n lines to the pick buffer.

ARG *motion* PICK, where the argument is cursor defined, causes the indicated text to be copied into the pick buffer. If lines are indicated, the effect is the same as for ARG n PICK. Indicated areas are placed in the pick buffer as partial lines, and will be inserted by PUT within lines, pushing to the right any text to the right of the insertion (see next section).

Anything placed into the pick buffer is also placed in the # file (see Sec. 13.2).

PICK functions do not alter the screen or the files being edited.

11.5. Placing Copies of Text in the File: PUT

Pressing the function key marked PUT places the contents of the pick buffer at

the current cursor position. ARG PUT places the contents of the close buffer at the current cursor position. The contents of the buffer are not affected.

A buffer may contain lines of text, or an area of text. If the contents of the buffer were placed there by a cursor-defined argument which indicated a rectangular area of text, then the buffer contains a rectangle of text. Otherwise, it contains lines. (The contents of the buffer were placed there by either the PICK or the CLOSE function, Secs. 11.3 and 11.4.) The contents are inserted into the file in different ways, depending on whether they are lines or an area of text.

If the buffer contains lines of text, those lines are inserted starting at the current line, and subsequent lines are moved down to make room for the inserted lines.

If the buffer contains a rectangle of text, the rectangle is inserted with the upper left-hand corner of the rectangle at the current cursor position. The portions of the affected lines to the right of and including the current cursor column are moved to the right to make room for the inserted text. This may cause text to be moved beyond the right margin of the window. In this case, the text will no longer be visible, but the right margin character on lines containing such text will be changed from "|" to ">" to indicate that there is text beyond the right margin.

Two uses for PICK and PUT of rectangles of text are: (1) repeatedly inserting one or several words within lines as a rectangle of text one line high, and (2) repeatedly opening areas of text by putting in an area of blanks. The latter could also be accomplished by OPEN with a cursor-defined argument, but this takes many more keystrokes and the argument must be redefined for each area to be opened.

PUT may be used, in conjunction with PICK or CLOSE, to move text from one file to another, since the contents of the pick and close buffers remain unchanged when windows or current files are switched (see CHWIN, Sec. 11.12, and USE, Sec. 11.9).

11.6. Leaving the Editor: EXIT

EXIT causes termination of the editing session. All files which have been altered during the session are saved. The previous version of the file is renamed by appending the characters ".bak" to the old file name, and then truncating the name to 14 characters. If the old file name was 14 or more characters long, no ".bak" file is created. If an old ".bak" file existed, it is lost.

Files which have not been altered are not written out. However, files are considered changed if any printing characters were typed while editing the file, even if they did not result in an alteration of the file's contents. Since space is a printing character, using space instead of the move right cursor positioning key may result in a file being saved when the intent was merely to examine it.

ARG EXIT exits the editor as above, and then executes the most recent UNIX *load, compil,* or *ex* command. This is useful for programmers who wish to repeatedly edit and compile a program being debugged.

ARG a EXIT aborts the current editing session. No files are changed (unless explicitly saved earlier in the session by SAVE), and no ".bak" files are lost. The editing performed in the session may be recovered by manipulation of the keystroke file if desired (see Sec. 13.3).

ARG a dEXIT aborts the current editing session and provides a dump of the internal structures of the Editor. It is useful to system programmers debugging or changing the Editor.

ARG *stg* EXIT, where *stg* is anything but "a" or "ad", including a cursor-defined argument, has the same effect as ARG EXIT.

There is no way to abort or terminate the editing session other than by using the EXIT key when the Editor is ready to accept a command, short of killing the editor program from another terminal. Consult an experienced user should this become necessary.

11.7. Searching for a Text: +SCH and −SCH

ARG *string* +SCH, where *string* is any sequence of printing characters, searches forward in the file being edited for the next occurrence of the string. The search begins at the character position following the current cursor position. If the search string is found, the cursor is moved to the first character in that occurrence of the string. If that position was in the window before the search began, the cursor is simply moved there. If the position was not in the window, the window is moved down the screen to bring the line containing the string to a position about one-fourth of the way down from the top of the window.

When the cursor is moved to the located string, a bullet is flashed at that position for one second to make its location more apparent.

If the string does not occur between the current cursor position and the end of the file, the message "Search key not found" is displayed, and the window and cursor positions are unchanged.

+SCH, with no argument, will use the previous search string. Thus, to search to the second occurrence of a string one might type ARG *string* +SCH +SCH. If no prior search has been made during the editing session, an error message is displayed.

ARG +SCH will use the text in the window beginning at the current cursor position, and ending at the first blank, as its search string.

−SCH behaves similarly to +SCH, but −SCH searches backward in the file being edited, starting at the position preceding the current cursor position.

A search operation cannot be interrupted once it has been initiated. This can be annoying when searching in a long file.

When a search operation locates the search string in a long line of text, in a column past the right-hand boundary of the editing window, the window is moved right (see Sec. 11.14) in order to display the located string. This can be confusing, since most of the text will disappear from the screen. The left-hand border of the window will be displayed using the character "<" to indicate that there is text to the left of the border. This situation will not occur often, since normal editor use will not encounter lines longer than the width of the window.

11.8. Moving to a Specified Line: GOTO

ARG n GOTO, where n is a positive integer, moves the window so that line number n of the current file is on the top line of the window. The cursor is positioned

about one-fourth of the way down from the top of the window; the cursor will *not* be positioned on line n of the file.

GOTO without an argument is equivalent to ARG 1 GOTO. It moves the window to the beginning of the file. ARG GOTO moves the window to the end of the file. The last line of the file is displayed about one-fourth of the way down the window.

11.9. Editing and Creating Files: USE

ARG *filename* USE (USE is CTRL-B) causes the file named "filename" to become the current file being edited in the window. The previous file, window position, and cursor position are saved as the *alternate file* state for the window. Any previous alternate file state is lost.

If no file exists with the specified name, the Editor will display the message "Hit USE (CTRL-B) to make: filename." Pressing USE at this point will create a new file with that name, and use it as the current file; the previous file becomes the alternate file for the window. Pressing anything else instead of USE will return the editor to the previous state. No new file is created and the window and alternate file status are not altered.

USE, with no argument, causes the current and alternate file states to be switched. The window displays the alternate file. The window position in the file and the cursor position are restored to where they were when the alternate file state was recorded. The current file, window position, and cursor position are saved as the new alternate file state.

The alternate file state may be used to alternately edit two files. However, if the current file name is used again as the argument to ARG *filename* USE, then that file becomes both the current and the alternate file. In this case, USE is employed to switch back and forth between two different positions in the file without losing either of the window or cursor positions. This can be useful when two portions of the same file are being changed, for moving lines from one part of a file to another, etc. If lines are added or deleted in the current window at a point above the alternate window position, the alternate window position will be adjusted to point to the same text lines as before, even though those lines may now have a different line number in the file.

Pieces of text can be picked from one file and put into another by using the PICK, PUT, and USE functions in combination.

Normally, the Editor can be exited and restarted without losing the currently active file, window position, and cursor position. The command "ned !" may be used to restart Ned in a way that preserves all windows and all current and alternate file states (see Chap. 6).

11.10. Saving Files without Exiting: SAVE

Normally, a file is not changed on disk until the end of the editing session. Before that, the changes are recorded on the Editor's internal image of the file. Sometimes it may be desired to write the altered file on disk and continue with the editing session.

Pressing the function key marked SAVE (SAVE is CTRL-V) causes the file being edited to be written out on disk. The editing session may then be continued. The state of the editor is not changed. The previous version of the file is renamed to a new name generated by appending the characters ".bak" to the old file name, and then truncating the name to 14 characters. If the old file name was 14 or more characters long, no ".bak" file is created and the previous version of the file is lost. If an old ".bak" file existed, it is lost.

The file will be saved again upon exiting if it was altered at any time during the editing session, even if it was not altered again after the SAVE (see Sec. 11.6).

ARG *filename* SAVE will save the current image of the file being edited on a new file named "filename". If a file by that name already exists, it is renamed to a new name generated by appending the characters ".bak" to the old file name, and then truncating the name to 14 characters. If the old file name was 14 or more characters long, no ".bak" file is created and the previous version of the file is lost. If an old ".bak" file existed, it is lost.

ARG SAVE is similar to ARG *filename* SAVE, but takes its file name from the text in the window, starting at the current cursor position, and ending with the first blank.

If ARG *filename* SAVE is used to write on an alternate file or a file being displayed in another window, that file's image on disk will be altered, but the Editor will continue to display the original version of the file until the end of the session. If that file is subsequently saved by SAVE or by exiting, a new disk image of the file will be created, reflecting what the Editor thought the file was originally, or what it was changed to by editing. The ".bak" file may reflect the result of the ARG *filename* SAVE.

Saving a file being edited may provide some security in the event of a system malfunction, since it saves the results of all editing on that file up to the time of the save. However, it will invalidate the keystroke file, which can then no longer be relied upon for recovery (see Sec. 13.3).

11.11. Creating More than One Window: WIN

The screen initially consists of one large editing window. If it is desired to edit more than one file, or display different parts of a file simultaneously, the screen may be divided into up to ten smaller windows, each of which has its own current and alternate file, window position, and cursor position.

At any time, one window, called the *current window* or *active window*, contains the cursor and can be used to edit the current file being displayed in that window. The cursor cannot be positioned outside the current window by the cursor positioning keys, but the CHWIN function (Sec. 11.12) may be used to activate another window as the current window. The boundaries of the current window will be drawn using the standard margin characters. The boundaries of all inactive windows are drawn using the character ".".

A new window is created by dividing an existing window into two smaller ones. This is done by positioning the cursor next to one of the boundaries of the window to be divided, and pressing WIN (CTRL-Z). The window is divided by a new margin or boundary line extending through the cursor position, perpendicular to the adjacent boundary. The new window is the one below the new boundary, in the case of

a horizontal division, or the one to the right, if the division is vertical. The new window becomes the current one, and starts off with the system default file, /re.std, as the current file. Another file may be edited in the new window by the USE function (Sec. 11.9). The remainder of the old window continues to display its current file, and may be reactivated by the CHWIN function (Sec. 11.12).

A window can be created with a file initially displayed by typing ARG *filename* WIN.

If an attempt is made to create a new window when the cursor is not positioned next to an old window boundary, or when the cursor is in a window corner, the Editor will not know where the new boundary is to be placed, and will display the error message "Can't put a window there."

An interesting condition arises when the same or different portions of one file are being displayed in two or more windows. If a line is displayed in more than one window, any change made in that line is reflected in each of these windows. The inactive windows are updated each time the cursor is moved to a new line. Moreover, if lines are added or deleted in the current window at a point in the file before the portion displayed in another window editing the same file, the position of the other window will be adjusted to point to the same text lines as before, even though those lines may now have a different line number in the file.

Windows may be deleted, but only in the reverse order of their creation. To delete the most recently created window, and return its area to the window from which it was divided, type ARG WIN. It occasionally happens that, after a window is deleted, a boundary of that window will remain drawn in periods rather than the standard margin characters. This is a "cosmetic" malfunction, and does not affect the operation of the editor.

If an editing session in which there are multiple windows is terminated, the state of the Editor, including all windows, files, window and cursor positions, and alternate file states, may be restored by the UNIX command "ned !" (see Chap. 6).

11.12. Changing Windows: CHWIN

CHWIN (CTRL-C) is used when the screen contains more than one editing window. It activates the next window (in order of creation) to be the current editing window. The activated window has its margin drawn in the standard margin characters to indicate that it is active, and the cursor is restored to its previous position in that window. The alternate file status for that window is the same as it was when that window was last active.

CHWIN may take a positive numeric argument. In this case, the window with that number is activated. The windows are numbered in order of creation, starting with 1.

See also WIN (Sec. 11.11).

11.13. Setting and Clearing Tab Stops: S/RTAB

Up to a maximum of 20 tab stops may be set and cleared by the user. Tab stops are initially set at every eighth column in the file, beginning with column 9.

Pressing the function key marked S/RTAB (CTRL-[) causes a tab stop to be set at the column in which the cursor is positioned. ARG S/RTAB causes the tab stop in the cursor column, if any, to be cleared. Any string or cursor-defined argument to S/RTAB is ignored, and the tab stop in the cursor column is cleared.

Tab stops are set relative to the columns of a file, not the left margin of the window. Thus, if a file is moved left in a window (Sec. 11.14), the tab stops still correspond to the same positions in the text.

The same set of tab stops is used for all windows on the screen.

11.14. Editing Wide Lines: LEFT and RIGHT

In order to edit files which contain lines longer than the width of the window, the window may be moved left and right on the file, just as it is moved up and down. The RIGHT (CTRL-S) key moves the window 16 columns to the right. The left margin characters are replaced by the character "<" to indicate that there is text beyond the margin. The LEFT (CTRL-A) key moves the window 16 columns to the left, or to the left margin of the file, whichever is closer. If the window is repositioned at the left margin of the file, the margin characters are replaced by "|".

The cursor remains on the line it occupied prior to the move. If its original column is still on the screen, the cursor remains there. If the column has been moved off the screen, the cursor is moved to the column next to the margin nearest the original column.

The window may be moved a specified number of columns in either direction by ARG n LEFT and ARG n RIGHT, where n is a numeric argument.

Chapter 12

INTERACTIVE TEXT PROCESSING

Several text processing operations, such as substring replacement, paragraph fill and justification, indenting, and multiple spacing, are performed by programs which are run from within Ned by means of the EXEC (CTRL-X) key. The requested operation is performed on the indicated paragraphs or lines and the results displayed in the editing window. The mechanism by which this is implemented allows other operations to be added without further editor modification, both on a system-wide and on an individual user basis.

Section 12.1 gives a brief introduction to the EXEC function, to allow use of many of the capabilities without further explanation. Section 12.2 explains in more detail the arguments and effect of the EXEC key. Sections 12.3 through 12.6 describe more fully the most useful text processing functions available through EXEC. Section 12.7 gives details of the implementation of this mechanism for those wishing to provide new text processing modules for use with EXEC, and Sec. 12.8 provides examples of the way in which a wide range of text processing programs, or filters, available in the UNIX operating system may be useful in editing using EXEC.

12.1. Fill, Justify, and Replace: A Quick Introduction

The fill, justify, and replace features are all invoked by the EXEC key. For example, to fill (i.e., align the line lengths so that they are approximately the same) and justify the right margin of the paragraph of text starting at the line on which the cursor is currently positioned, type

```
ARG just EXEC
```

"Paragraph of text" means a sequence of lines separated by one or more blank lines.

To fill (but not justify right margins) the next 6 paragraphs, type

```
ARG 6 fill EXEC
```

To replace all occurrences of the word "Raquel" with the string "Ms. Welch" in the next 15 lines, type

```
ARG 15l rpl Raquel "Ms. Welch" EXEC
```

(The quotes are necessary since the string "Ms. Welch" contains a space.)

This is probably all you need to know in order to make use of these operations. The following sections give more detail concerning these and other operations that can be performed using the EXEC key.

12.2. Arguments and Effect of EXEC

The EXEC function is unique in that it takes more than one argument. As in UNIX commands, the arguments are separated by one or more spaces. The arguments are:

1. An optional first argument, telling the number of lines (negative integer) or paragraphs (positive integer) to be operated on. If this argument is omitted, it is assumed to be 1 (one paragraph). Paragraphs are groups of non-blank lines separated by one or more blank lines. The first line to be operated on is always the line currently containing the cursor. If lines past the end of the file are specified, only lines up to the end of the file are used. Thus a large argument, such as 999, may be used to specify the entire file, starting with the current line. A number of lines may alternatively be specified by an argument consisting of a negative integer, or equivalently, an integer followed by the letter "l". For example, −15 and 15l both specify a block of 15 lines.
2. The name of the operation (filter) to be performed.
3. Arguments to the operation. These vary with the individual operation. If an argument contains one or more spaces, it is necessary to enclose it in paired single or double quotes (' ' or " ").

The effect of the EXEC function key is to feed the lines to be operated on to the function program specified, and then replace them by the same lines as processed by that program. The old lines are placed in the close buffer and the previous contents of the close buffer are lost.

12.3. The Replace Function: Rpl

The program *rpl* requires two arguments. It replaces all occurrences of the character string which is its first argument by the string which is its second argument. Quotes must be used to enclose an argument containing spaces.

Rpl recognizes regular expressions in its first argument. For the definition of regular expression, see ed(I) in the UNIX Reference Manual (except that rpl uses " @ " instead of "." to match any character). Thus, to delete all sequences of blanks at the beginning of the next 100 lines, type

```
ARG 100l rpl "^ *" "  " EXEC
```

(Note that this will not delete tabs at the beginning of lines.) This means that the characters ^ , $, @, *, [, and] must be preceded by " \ " to be recognized in the first argument of rpl.

To delete a string using rpl, replace it by the null string " ".

12.4. Justifying Text Paragraphs: Just

The program *just* passes its text lines to nroff to be filled and justified. (Nroff is a text formatting program in the UNIX operating system; see "NROFF Users' Manual" by Joseph Ossanna, in *Documents for Use with the UNIX Time-Sharing*

System, Sixth Edition, Bell Telephone Laboratories.) Options for just are identical with those for nroff, except that hyphenation is normally turned off, all escape characters are set to the tilde character (so as not to alter input text unexpectedly), and paging is disabled.

To deal correctly with indented paragraphs, just always indents the first and second lines of a paragraph exactly as they are indented in the source text, and lines up all subsequent lines with the second line. The special treatment of indented paragraphs may be suppressed; see below. All multiple spaces and tabs within a line are treated by just as single spaces. The effect is to allow the user to correct already-justified paragraphs, and then rejustify the text. However, this means that multiple spaces will be lost in text passed through just.

It is also possible to enter a paragraph of raw text, in free format, starting the first two lines at the proper indentations, and then use just to justify the entire paragraph.

One or more arguments may be given to just If the argument begins with a ".", it is taken to be an nroff command line and passed directly to roff. (If the argument does not begin with ".", it is given to nroff as an argument.) Arguments which alter nroff escape characters will have no effect, however.

Example: to justify the next three paragraphs with hyphenation:

```
ARG 3 just ".hy 1" EXEC
```

Example: to justify the next 20 lines between columns 10 and 70, pass to nroff the .in (indent) and .ll (length) commands, as follows:

```
ARG 20l just ".in 10" ".ll 70" EXEC
```

If one of the arguments to just is the single character "x", just will not attempt to process indented paragraphs correctly or remove multiple spaces before passing the text to nroff. Nroff will receive the text exactly as it appears in the source file.

12.5. Filling Text Paragraph: Fill

Fill works identically to *just* (Sec. 12.4) except that right margins are not justified. The effect of fill is to make all lines approximately the same length.

12.6. Multiple Spacing and Indentation: Space

The program *space* manipulates paragraphs of text by double spacing, multiple spacing, and/or indenting the lines. It takes either zero, one, or two arguments. With no arguments it will double-space text. The argument $-n$, where n is an integer, will cause the text to be n-spaced; i.e., $n-1$ blank lines will be inserted between every two lines of text. The argument $-in$, where n is an integer, will cause each line to be indented n spaces. Thus, to double-space a paragraph, type

```
ARG space EXEC
```

To single space two paragraphs and indent them ten spaces, type

```
ARG 2 space -1 -i10 EXEC
```

12.7. How EXEC Works

This explanation of the EXEC mechanism is presented for readers who have a detailed knowledge of the UNIX operating system.

The EXEC key causes the Editor to spawn a fork with a pipe as standard input and the editor temporary file, positioned at the current end of file, as standard output. The fork then runs the named program as a filter, and dies. The Editor examines the termination status of the fork. A status of -1 is reserved for "File not found" when trying to execute. A status of -2 causes a "Program terminated abnormally" diagnostic. If the status is anything else, whatever was written on the standard output by the filter is inserted in place of the input lines. The removed lines, if any, are placed in the close buffer, and may be retrieved by ARG PUT.

The named filter is searched for in the same manner as the shell would: working directory, user's /bin, /bin, /usr/bin. Thus users may write their own editor filters, and may override system filters of the same name.

12.8. More General EXEC Capabilities

The UNIX operating system contains a wide variety of text processing programs suitable for use with EXEC. For information on specific programs, see the *UNIX Programmer's Manual* by K. Thompson and D. Ritchie. Any program suitable for use as a filter may be used with EXEC. Thus, to pick five lines onto file "file1," type

```
ARG 5l tee file1 EXEC
```

This replaces the five lines with their output from tee—i.e., the five lines are not changed—but tee writes them on file1 as well.

To insert file "file2", type

```
ARG 0 cat file2 EXEC
```

This inserts (replaces 0 paragraphs) file file2 (the output of "cat file2") at the current line.

To sort the lines in a paragraph in alphabetical order, type

```
ARG sort EXEC
```

To insert the current date and time at the current line, type

```
ARG 0 date EXEC
```

To replace a line consisting of an arithmetic expression (e.g., 32.7 * sin(2.68)) by its value, type

```
ARG bas EXEC
```

(thus invoking the BASIC language processor to compute the expression).

The uses of the EXEC key are limited only by the ingenuity of the user.

Chapter 13

RECOVERY

The Editor contains several backup features to prevent destruction of information by system failures or by inadvertent user actions.

13.1. Old Versions of Files: ".bak" Files

When an altered file is saved, either by exiting at the end of an editing session, or by the SAVE function (Sec. 11.10), the previous version of the file is not deleted, but is renamed to a new name generated by appending the characters ".bak" to the old file name, and then truncating the name to 14 characters. If the old file name was 14 or more characters long, no ".bak" file is created, and the old version of the file is lost. If an old ".bak" file existed, it is lost.

Administrative procedures at an individual computer installation may result in the periodic deletion of ".bak" files in order to save disk space. At the time of this writing, the procedure on the Rand-UNIX system is to delete ".bak" files every night. Note, however, that files whose names are 11 to 13 characters in length will have backup files ending in ".ba", ".b", or ".", since the backup file name is truncated to 14 characters. These files may remain in existence indefinitely.

13.2. Recovering Deleted Text: The # File

When text is deleted using the CLOSE function, the most recently deleted material is saved in the close buffer (Sec. 11.4). Previously deleted material is no longer available from the close buffer. However, the material may be recovered at any time prior to the end of the editing session through a mechanism known as the "# file".

The # file behaves just like a file with the name "#", except that it is completely internal to the Editor. It is never written out on disk, and disappears at the end of each editing session. Each time material is inserted into either the pick buffer or the close buffer, the same material is inserted at the end of the # file. This process can be viewed directly by creating a new window containing the # file (the command is ARG # WIN; see Sec. 11.11), and then picking or deleting text using PICK or CLOSE. The text will appear simultaneously in the # file.

If it is desired to recover any material that has been deleted during the current editing session, the # file may be edited using the USE (Sec. 11.9) or WIN (Sec. 11.11) commands with the file name "#". The text to be recovered may be located in the # file and moved to the file being edited using PICK and PUT (Secs. 11.4 and 11.5).

13.3. Recovering from Crashes: The Keystroke File

Every keystroke typed during an editing session is saved on a special *keystroke file* to enable the session to be reproduced. This is desirable for two reasons. First, if the system malfunctions before the Editor is exited and the updated files saved, reproducing the session up to the point of the system failure would avoid losing the work done up to that point. Second, if a particular sequence of user actions produces an editor malfunction, the ability to reproduce that precise sequence of actions will enable system software personnel to identify and repair the problem.

The name of the keystroke file is /tmp/rettyX.fred, where ttyX is the name of the terminal in use, and fred is the user's login name. (The name and terminal of all users is displayed by the UNIX "who" command.) This file is reused every time Ned is run. Therefore, if the keystroke file is to be used, it must be saved under another file name before using the Editor again from the same terminal. To save the keystroke file, execute the UNIX command "mv /tmp/rettyX.fred newname", where "newname" is a previously unused file name in your directory, and the terminal and login name actually in use are substituted for ttyX and fred.

To reproduce the editing session exactly as it occurred, the keystroke file should be saved as described above. Also, the files edited in that session should be restored to their state before the start of that session. For example, if file "fname" was edited, and the session terminated normally, the old fname is now fname.bak, and a new fname has been made. If it is desired to reproduce the session, fname.bak must be restored to fname by the UNIX command "mv fname.bak fname". If fname has not been changed, there is no need to put it back. If, however, fname were saved during the session by the SAVE function, and then again at the end of the session by exiting normally, the original file fname has by now been first renamed to fname.bak, and then lost entirely at the second save. Thus the session could not be reproduced.

If an editing session is initiated by the command "ned", with no arguments, and terminates normally, it generally cannot be reproduced, since the initial editor state will be lost. However, if such an editing session is terminated abnormally, such as by an editor or system crash, the session can be recovered *as long as the recovery is the first use of the Editor by the user after the crash.*

Once the keystroke file has been saved and the files that were changed have been restored to their previous state, the editing session may be reproduced by the UNIX command "ned-ttyfile", where ttyfile is the name of the saved keystroke file. The Editor will perform all the actions of the previous session. This is worth doing once just to watch.

If the session was terminated by a system crash, the Editor will come to the end of the keystroke file and pause in its current state. At this point, you may proceed with the editing session by using the keyboard.

If the session was terminated by an editor crash, the crash will be reproduced, together with a dump of the internal data structures of the editor. If the session was terminated normally, the reproduction of that session will also terminate normally.

Often it is not desirable to reproduce the entire session. If the last keystroke caused the Editor to crash, it is more useful to reproduce the session up to the next to last keystroke, and then try to save the edited files at that point.

The Editor may display an error message giving the cause of a crash. If the

cause was running out of space, it may be necessary to delete a number of keystrokes from the end of the keystroke file before the edited files can be saved without crashing the Editor.

The keystroke file contains many control characters, corresponding to the function keys typed during the editing session. As explained in Sec. 9.4, control characters in a text file are displayed by the Editor as a bullet followed by a printing character. To delete a control character, it is necessary to delete both the bullet and the following character.

The procedure for deleting the last few characters of the keystroke file is as follows: Edit the file using Ned. The many bullets on the screen are all the function and motion keys typed during the last editing session. Find the last line of the file by moving the window down the file while looking in the left margin. The last line in the file will have the left margin character "|", while all following lines will have the left margin character ";".

Now look at the right margin character on that line. If it is a ">", the line is too long to fit in the window, and it is necessary to move the window to the right using the RIGHT key (Sec. 11.14) until the right margin character becomes a "|". Leave the cursor on the last line in the file to mark it, since the left margin characters which identify it as the last line will be lost once the window is moved right.

Since lines in the keystroke file may be quite long, the window might have to be moved quite a distance, using RIGHT with a numeric argument. Using RIGHT, and LEFT too if necessary, move the window so that the last characters of this last line are on the screen. Then delete the last few bullet-printing character pairs, by spacing over them, and save the edited keystroke file by exiting from the Editor. This edited keystroke file should be able to reproduce the editing session, minus the last few disastrous strokes, and leave the editor ready for use from the keyboard at that point in the session.

Chapter 14

MISCELLANEA

14.1. Stopping the Editor in the Middle

Because the editor takes some time to update the screen following such commands as +PAGE, it is possible to type another command before the screen has been completely rewritten. In many cases, the editor will abort writing the screen and perform the next command. This is done whenever the editor senses that the screen will have to be rewritten again for the new command. For example, it is possible to press +PAGE several times in rapid succession without having to wait for the editor to fill the screen each time.

However, there are several commands, such as +SCH and EXEC, which may take a long time to perform. Since the keys which usually interrupt a program's execution are used differently in Ned, there is no way to abort a running command, or to terminate the editing session other than by using the EXIT key when the editor is ready to accept a command, short of killing the editor program from another terminal. Consult an experienced user should this become necessary.

14.2. The /re.std File

The system contains a text file called /re.std. It is the *default file* which is edited in several situations when the editor does not have a satisfactory file to look at. It contains a message to the effect that the file the user tried to look at does not exist. This can happen when Ned is run without arguments and attempts to determine the file edited in the previous session. If that information has disappeared (which may happen overnight), /re.std will be displayed.

14.3. Editor Limitations

The editor is subject to several limitations. It may be used to edit lines of any length, but cannot edit files containing more than 32,768 lines. An attempt to do so may cause the editor to crash.

No more than 20 tab stops may be set. No more than eight files may be examined or edited during an editing session. At most ten windows may be created. Exceeding any of these limitations will result in an error message being displayed.

Chapter 15

ERROR MESSAGES

This chapter provides further explanation of the error messages which may be produced by the editor, and suggests some actions to recover from the error situation.

An error message is displayed on the bottom line of the screen, and rings the terminal's bell (on terminals so equipped). The most frequent causes of error messages are inadvertently striking the wrong key, or trying to perform a function that makes no sense (such as deleting −3 lines).

Most of the error messages provided by the editor are listed below in alphabetical order, with explanations. Those error messages not listed are likely to indicate system or editor malfunctions, and, if they persist, should be reported to system personnel.

"Abnormal termination of program."

Cause: The program invoked by the EXEC key terminated in an error condition. Perhaps it was given a wrong number of arguments, or an improper argument. Perhaps the text specified to be processed was not acceptable to that program.

Action: Check the validity of the arguments and specified text.

"Argument must be a string."

Cause: You provided a cursor-defined argument, or a null argument, to a function expecting a number or a string.

Action: Redo the function providing an appropriate argument.

"Argument must be numeric."

Cause: You provided a non-numeric argument to a function expecting a number. For example, typing

```
ARG j +LINE
```

would cause the above error message to be displayed since the number of lines must be an integer.

Action: Redo the function providing a positive integer as an argument.

"Argument must be positive."

Cause: You provided a negative number as an argument to a function expecting a positive number. For example, typing,

```
ARG -2 OPEN
```

would cause the above error message to be displayed since the number of lines to open must be a positive integer.

Action: Redo the function providing a positive integer as an argument.

"Badkeyerr — editor error."

Cause: A function key was struck which has no meaning to the Editor.

Action: Hit another key.

"Can not fork or write pipe."

Cause: The EXEC command has encountered a system problem while trying to perform the requested function.

Action: Notify system software support personnel.

"Can't find program to execute."

Cause: The program specified to the EXEC function does not exist, or perhaps the argument to the EXEC command was in the wrong format.

Action: Check it out.

"Can't make any more ports."

Cause: You have tried to create more than 10 windows on the screen. The maximum number permitted is 10.

Action: Discontinue window creation or delete some of the windows before creating more (see WIN, Sec. 11.11).

"Can't put a window there."

Cause: You have placed the cursor in an unacceptable position for window creation.

Action: Reposition the cursor correctly (see WIN, Sec. 11.11).

"Can't write in specified directory."

Cause: You tried to save a file in a directory for which you do not have write permission.

Action: Check your file specification, or save the file in a directory for which you do have write permission.

"Can't write on pipe."

Cause: The EXEC command has encountered a system problem while trying to perform the requested function.
Action: Notify system software support personnel.

"DANGER — WRITE ERROR."

Cause: This could be caused by a computer hardware failure. But it could also be due to attempting to save on a file to which you do not have proper access.
Action: Check your file name specification. If it is correct, attempt the save again. If failure persists, notify system hardware support personnel.

"Directory does not exist."

Cause: You attempted to edit a file in a nonexistent directory.
Action: Check your file name specification.

"Feature not implemented yet."

Cause: You have pressed some combination of keys that the Editor does not recognize.
Action: Try something else.

"File read protected."

Cause: You tried to edit a file which exists, but which is protected so that you cannot read it.
Action: Change the protection of the file, or give up trying to edit it.

"First the bad news . . ."

Cause: This message indicates a fatal editor malfunction. The good news is that the work done in your editing session can most likely be recovered (see Sec. 13.3). If the message continues ".ran out of space.", then the problem was due to editing too large a file, too long a line, or too many long files, or to making many changes in large files. Otherwise, the problem is an editor bug. In this case, it is imperative to preserve the relevant files so that the bug can be reproduced and fixed.
Action: See "Recovering from Crashes," Sec. 13.3.

"Hit USE (CTRL-B) to make: filename."

Cause: You tried to edit a file that does not exist. It may have been an argument to the USE function, an

argument to the "ned" UNIX command when starting the editing session, or a file that was edited in a previous session but could not be found at the start of the current session because it had been deleted in the meantime or because the working directory had been changed, making its name invalid from the new directory.

Action: Hit USE to create a file by that name. Hit ARG to abort the attempt to edit that file (see USE, Sec. 11.9).

"Invalid argument."

Cause: The argument you typed is in the wrong format for the function button you invoked.

Action: See the appropriate section of Chap. 10 for the correct format for this function.

"Margin stuck; move cursor to free."

Cause: You have tried to type past the end of the line in view.

Action: Move the cursor or move the window (see RIGHT, Sec. 11.14).

"Nothing in the CLOSE buffer."

Cause: You have pressed ARG PUT when no lines have been closed.

Action: Use CLOSE to remove appropriate lines and repeat the ARG PUT.

"Nothing in the PICK buffer."

Cause: You have pressed PUT when no lines have been picked.

Action: Use PICK to identify the lines to PUT.

"Nothing to search for."

Cause: You have pressed a +SCH or a −SCH without having specified a string to search for.

Action: Provide a string as an ARG to a +SCH or a −SCH.

"Printing character illegal here."

Cause: You typed a printing character while defining a cursor-defined argument. Only cursor positioning keys (Chap. 8) are legal here.

Action: If defining a cursor-defined argument, proceed. If you want to type a numeric or string argument, press ARG once to abort the current argument and again to start the argument over (see Sec. 10.1).

"Search key not found!"

Cause: The string you specified as an ARG to a +SCH does not occur between the current cursor position and the end of the file. For a −SCH the string does not occur between the current cursor position and the beginning of of the file.
Action: No corrective action required.

"Specified directory does not exist."

Cause: You attempted to save a file in a nonexistent directory.
Action: Check your file name specification.

"Too many files — Editor limit."

Cause: You have attempted to edit more than eight files during an editing session.
Action: EXIT, then run the Editor again using the command "ned !". This will restore all windows, files, and alternate files just as before, but if there are eight or fewer active and alternate files the limit will not be exceeded.

"Too many tabstops; can't set more."

Cause: You have attempted to set more than 20 tab stops, including the 10 default stops (unless they were cleared).
Action: Clear some tab stops before attempting to set more (see Sec. 11.13).

"You cannot modify this file."

Cause: You have attempted to alter a file which is write protected.
Action: Cease attempts to change the file.

Appendix A

SUMMARY OF NED FUNCTION KEYS

KEY	Sec. for Ref.	Action of KEY	Action of ARG KEY	Action of ARG X KEY (X = any argument)[a]	Action of ARG Position KEY (Position = cursor-defined argument)
+PAGE -PAGE	11.1	Moves window forward (+) or backward (-) one page	ERROR[b]	X is numeric. Moves window forward (+) or backward (-) X pages.	ERROR
+LINE -LINE	11.1	Moves window forward (+) or backward (-) about ¼ page.	Moves window so CCL[c] is the first line in window (+) or last line in window.	X is numeric Moves window forward (+) or backward (-) X lines.	ERROR
OPEN	11.2	Inserts a blank line above CCL.	Moves the part of CCL starting with cursor position to a new next line.	X is positive Inserts X blank lines above CCL.	Inserts blank lines or rectangles in area defined by cursor.
CLOSE	11.3	Deletes CCL and places it in CLOSE buffer.	Deletes the part of CCL starting with cursor position replacing it with the line below CCL.	X is positive. Deletes X lines starting with CCL and puts them in CLOSE buffer.	Deletes lines or rectangle defined by cursor and puts it in CLOSE buffer.
PICK	11.4	Places CCL in PICK buffer.	ERROR	X is positive Places X lines starting with CCL in PICK buffer.	Places lines or rectangle defined by cursor in PICK buffer.
PUT	11.5	Places contents of PICK buffer at cursor position.[d]	Places contents of CLOSE buffer at cursor position.[d]	ERROR	ERROR
EXIT	11.6	Exits Editor. All altered files are written to disk; old versions are renamed "name.bak".	Exits Editor, saving files as in EXIT case, then executes most recent ex, load, or compil UNIX command.	If X is the letter a, exits without writing any files; all editing is lost and files are unchanged	Exits Editor, saving files as in EXIT case, then executes most recent ex, load, or compil UNIX command.
+SCH -SCH	11.7	Searches forward (+) or backward (-) for the last string searched for (if any).	Searches forward (+) or backward (-) for the string pointed to by the cursor up to the first blank.	Searches forward (+) or backward (-) for the string X.	ERROR

[a]"X is numeric" means the argument must be a signed integer. "X is positive" means the argument must be an unsigned integer. If one of these restrictions applies, and an argument violates it, an error message will be given.

[b]"ERROR" means an error message will be given if this function sequence is typed to the Editor.

[c]CCL is the Current Cursor Line.

[d]If buffer contains full lines, CCL and following lines are pushed down to make room for inserted lines. If buffer contains a rectangular area, its contents are inserted with the cursor position at the upper left-hand corner of the rectangle, and partial lines to the right of the inserted material are pushed to the right.

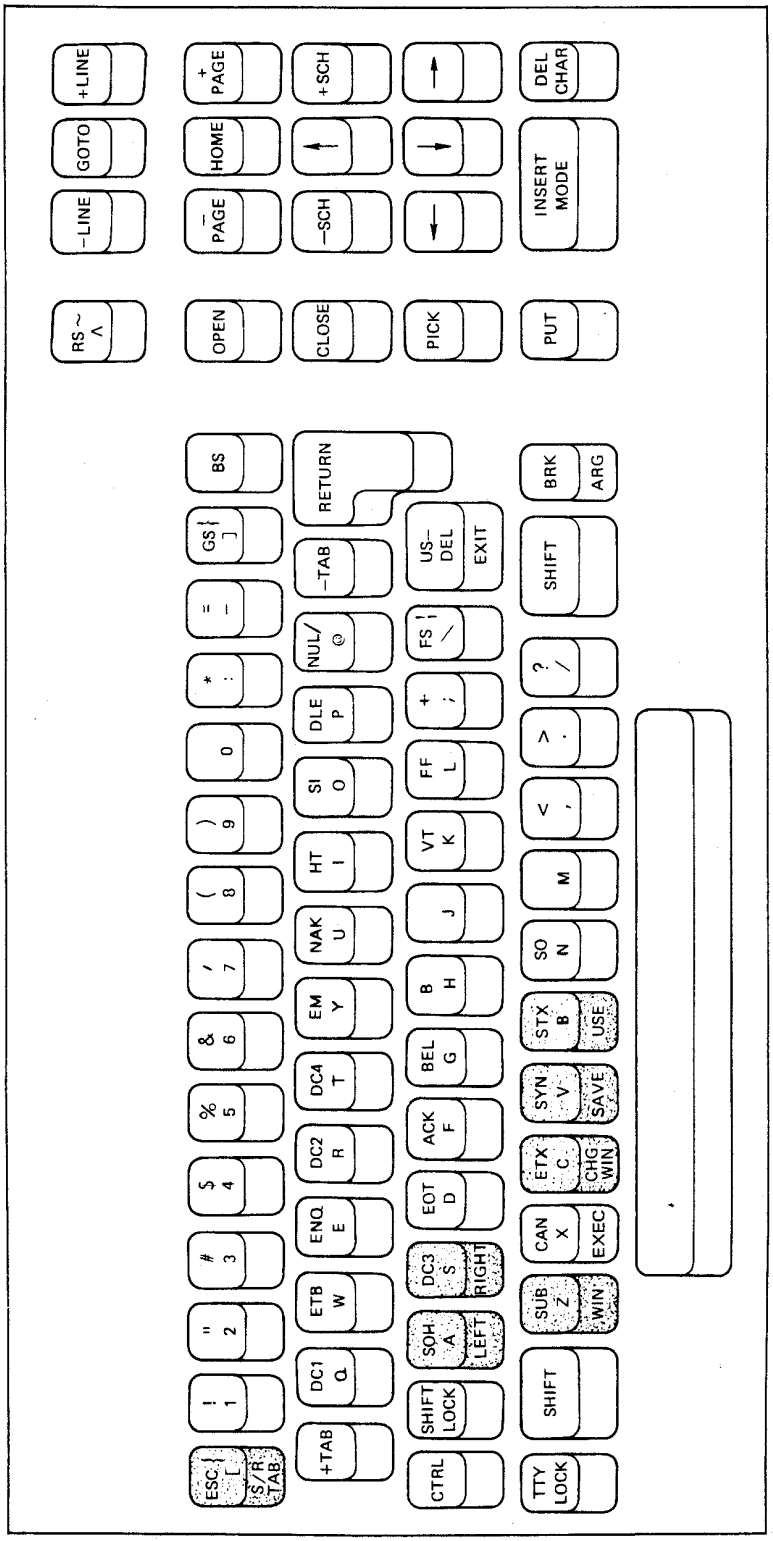
Summary of NED Function Keys (continued)

KEY	Sec. for Ref.	Action of KEY	Action of ARG KEY	Action of ARG X KEY (X = any argument)[a]	Action of ARG Position KEY (Position = cursor-defined argument)
GOTO	11.8	Moves window so top line is at line 1 of file.	Moves window so the last line is about ¼ of the way down the window.	X is numeric Moves window so top line is at line X of file (or 0 if X<0).	ERROR
USE (CTRL-B)	11.9	Switches window to alternate file; current file becomes alternate file.	ERROR	Edits file with name X as current file; old current file becomes alternate file.	ERROR
SAVE (CTRL-V)	11.10	Writes current file out on disk.	Writes current file out on disk on file name taken from cursor position up to next blank	Writes current file out on file named X.	ERROR
WIN (CTRL-Z)	11.11	Makes a new window with border extending from cursor position (which must be next to a border).	Deletes last window created.	Makes a new window and edits file X in it.	ERROR
CHWIN (CTRL-C)	11.12	Switches current window to be next window.	ERROR	X is numeric. Switches current window to be Xth created window.	ERROR
S/RTAB (CTRL-[)	11.13	Sets a tab stop at cursor column.	Removes any tab stop at cursor column.	Removes any tab stop at cursor column (X is ignored).	Removes any tab stop at original cursor column (ARG is ignored).
EXEC (CTRL-X)	12	ERROR	ERROR	X is a command in the format [n] prg [arg...] (in UNIX notation). Replaces n paragraphs (or -n lines if n >0) by result of running filter program on that text with given agruments. Old paragraphs are placed in the CLOSE buffer.	ERROR
LEFT	11.14	Moves window left 16 columns or to file boundary, whichever is less.	ERROR	X is numeric Moves window left X columns or to file boundary, whichever is less.	ERROR
RIGHT	11.14	Moves window right 16 columns.	ERROR	X is numeric. Moves window right X columns.	ERROR

[a]"X is numeric" means the argument must be a signed integer. "X is positive" means the argument must be an unsigned integer. If one of these restrictions applies, and an argument violates it, an error message will be given.

Appendix B

DIAGRAM OF TERMINAL KEYBOARD



Appendix C

INDEX OF CAPABILITIES

This appendix lists the various editing operations that can be performed by Ned. Where appropriate, brief instructions are given for performing the operation, and a page reference in this manual is provided for a more detailed explanation.

- To abort an editing session: use ARG a EXIT. See p. 24.
- To close up a rectangle of text in the middle of adjacent lines: use CLOSE with a cursor-defined argument. See p. 23.
- To create a new file: use USE. See p. 26.
- To create and delete multiple editing windows: use WIN. See pp. 27-28.
- To delete characters from the middle of a line: use DELCHAR, or use BS in insert mode. See p. 17.
- To delete lines: use CLOSE. See pp. 22-23.
- To duplicate an editing session: run Ned on a keystroke file. See pp. 35-36.
- To edit an argument while typing it: use only BS to erase characters, or ARG to abort it entirely. See pp. 19-20.
- To edit different files: use USE. See p. 26.
- To erase characters: type over them with the space bar or BS. See pp. 8, 17.
- To fill and/or justify one or more paragraphs: use fill or just. See pp. 30-32.
- To flip back and forth between two files or between two places in the same file: use USE. See p. 26.
- To get back text that has been deleted accidentally with CLOSE; use ARG PUT or the # file. See p. 34.
- To indicate an area of text for OPEN, CLOSE, or PICK: use a cursor-defined argument for the function key. See p. 20.
- To initialize Ned at other than line 1 of a file: see p. 14.
- To insert a file into the file being edited: use cat with EXEC. See p. 33.
- To insert blank lines: use OPEN. See p. 22.
- To insert characters into a file: position the cursor at the desired location and type the characters. See p. 8.
- To insert one or more characters in the middle of a line: use insert mode. See pp. 8, 17.

- To make copies of one or more lines: use PICK. See p. 23.
- To modify a function key by an argument: type ARG, the argument, and the function key. See pp. 19-20.
- To move text between files: use PUT together with USE or multiple windows. See p. 24.
- To move the cursor rapidly in a horizontal direction: use +TAB and −TAB. See p. 16.
- To move the cursor rapidly in a vertical direction: type ARG, a number, and a vertical positioning key. See p. 16.
- To move the window so the current line is at the top or bottom of the screen: use ARG +LINE or ARG −LINE. See p. 21.
- To move the window up and down the file: use +PAGE, −PAGE, +LINE, −LINE, and GOTO. See p. 21.
- To open up a rectangular area in the middle of several adjacent lines: use OPEN with a cursor-defined argument (or PUT of a blank rectangle). See p. 22.
- To position at a particular line of a file, or at the first or last line: use GOTO. See p. 25.
- To position the cursor: use the keys marked with arrows and the other cursor positioning keys. See p. 16.
- To put page ejects into a text file: type QUOT CTRL-L. See p. 18.
- To put the same word or phrase down in many places: see p. 24.
- To put two adjacent lines together to make one long line: use ARG CLOSE. See p. 22.
- To recover an editing session that ends in a crash: use the keystroke file. See pp. 35-36.
- To replace multiple occurrences of one string by another: use rpl. See p. 31.
- To restore a file to its state before the last edit: use the ".bak" file. See p. 34.
- To restore Ned with all windows from a previous session: use the UNIX command "ned !". See p. 14.
- To search for a character string: use +SCH and −SCH. See p. 25.
- To set or clear tab stops: use S/RTAB. See p. 28.
- To sort lines of text into alphabetic order: use sort with EXEC. See p. 33.
- To split a line in the middle, making two lines: use ARG OPEN. See p. 22.
- To terminate an editing session and save the files: use EXIT. See p. 24.
- To type control characters into a file: use the QUOT key. See p. 17.

• To type or edit past the right margin: move the window sideways using LEFT and RIGHT. See p. 29.

• To type text at the end of a file without repositioning the window: use RETURN at the end of each line, and +LINEs will be performed automatically at the bottom of the window. See p. 21.

• To write out an edited file under a different file name: use ARG *filename* SAVE. See p. 26.

REFERENCES

Irons, E. T., and F. M. Djorup, "A CRT Editing System," *Communications of the ACM*, January 1972.

Kelly, Jeanne, *A Guide to Ned: A New On-Line Computer Editor*, The Rand Corporation, R-2000-ARPA, July 1977.

Kernighan, Brian, "UNIX for Beginners," *Documents for Use with the UNIX Time-Sharing System*, Bell Telephone Laboratories, Murray Hill, N.J., undated.

Ossanna, Joseph, "NROFF Users' Manual," *Documents for Use with the UNIX Time-Sharing System*, Bell Telephone Laboratories, Murray Hill, N.J., September 1974.

Thompson, Kenneth, and Dennis Ritchie, *UNIX Programmer's Manual* (Sixth Edition), Bell Telephone Laboratories, Murray Hill, N.J., May 1975.

Weiner, P., et al., *The Yale Editor "E": A CRT Based Editing System*, Yale Computer Science Research Report 19, April 1973.

INDEX