Screen editor EDB83

Lynne Thompson  
*University of Wollongong*

Monika Strasser  
*University of Wollongong*

Rodney Porter  
*University of Wollongong*

Jennifer Tran  
*University of Wollongong*

Dao Trinh  
*University of Wollongong*

Follow this and additional works at: [https://ro.uow.edu.au/compsciwp](https://ro.uow.edu.au/compsciwp)

**Recommended Citation**

Thompson, Lynne; Strasser, Monika; Porter, Rodney; Tran, Jennifer; and Trinh, Dao, Screen editor EDB83, Department of Computing Science, University of Wollongong, Working Paper 84-1, 1984, 87p.  
SCREEN EDITOR EDB83

Lynne Thompson
Monika Strasser
Rodney Porter
Jennifer Tran
Dao Trinh

Preprint No. 84-1 March 7, 1984

Submitted in partial fulfillment of the requirements of CSCI 321
Software Project at the University of Wollongong in 1983

=============================================================
P.O. Box 1144, WOLLONGONG, N.S.W. 2500, Australia
Tel: (042) 270 859
Telex: 29022
INDEX


Part 2. User Guide

PART I

TECHNICAL REPORT

SCREEN EDITOR EDB83

TECHNICAL REPORT

Lynne Thompson
Monika Strasser
Rodney Porter
Jennifer Tran
Duo Trinh

Software Project CSCI321
Group B

University of Wollongong
October 1983
1. INTRODUCTION

The design of EDB33, a screen editor based upon R. Miller's EDIT, can be defined using the terminology introduced by J. Reinfield, in his sample design document, "The Design of a Simple Editor". This terminology is defined in the following "Definitions" section.

Any interactive program may be considered in terms of spaces, dimensions and mechanisms. An editor, in particular, consists of two major spaces, the user interface and the computer interface. Both of these spaces consist of the same three dimensions, but vary in regard to the mechanisms acting upon them.

In this document we will concern ourselves with the definition of the computer interface. The "physical" aspects of the user interface, the commands entered at the keyboard, have already been dealt with, in some detail, in the previously submitted system specification document, and will be fully documented in the EDB33 user's manual. The more abstract concepts, which deal with the user's conception of the system, vary with each user, and will be excluded from this discussion.

Also included in this document, is a series of hierarchy charts for the EDB33 system. These charts form a bridge from the design of the system in terms of mechanisms, to a more comprehensive view of the system's structure in procedural form.
1-1- Definitions

1-1-1. Space

- is an independent aspect of the whole problem and is based upon the given software and hardware equipment, and upon the specification of the task in hand. A space consists of one or more dimensions.

1-1-2. Dimension

- is a large component of the design which is (ideally) orthogonal to all other components, so that it communicates only loosely with other dimensions by simple messages or via a few global data structures. Each dimension is made up of one or more mechanisms.

1-1-3. Mechanism

- is a self-contained set of operations referring to one or more data structures in one or more dimensions. A simple mechanism is a mechanism contained in one dimension. A mechanism which affects more than one dimension is called a compound mechanism and a global mechanism affects all dimensions.

1-1-4. Operator

- is a procedure or function which performs a well defined task upon one or more data structures. An operator which affects only one mechanism is a primitive operator. A compound operator affects more than one mechanism.

1-1-5. Input file

- resides on a system disc and is the file specified by the user to be edited. There is no upper or lower limit on the size of the file, which consists of lines of characters, printable and non-printable, where each line is terminated by a newline character.

1-1-6. Edit file

- is the file on which the screen editor operates. Initially it is a copy of the Input file with any non-printable characters filtered out and replaced by printable characters, where appropriate. The edit file consists of a sequence of text lines, and it may reside in memory or on disc, or both, at any time during the editing process.
1.1.7. Text line

   - is a string of at most 255 printable characters. Any longer lines that exist in the input file are therefore truncated when the edit file is created.

1.1.8. Word

   - is a string of printable characters terminated by a blank space.

1.1.9. Non-printable character

   - is any character code which does not represent a printable symbol. The blank space is regarded as a printable symbol.

1.1.10. Page list

   - is a linked list of pages used to record the contents of each 512 byte block.

1.1.11. Frame

   - is a space defined in memory to hold a block of text so that the text may be manipulated as required. At least 2 are necessary for use by the screen editor.

1.1.12. Temporary file

   - is a file designed to store those blocks of text which do not fit into the frames. The blocks of text held in the frames, together with the temporary file form the edit file.

1.1.13. Free list

   - is a list recording the location of available empty space. The free list is used as the source when a request is made for additional space in the edit file.

1.1.14. Text window

   - is the top portion of the terminal screen, which displays up to 23 screen lines of the edit file. The text window is initially positioned over the top left portion of the edit file, and is defined by two pairs of text co-ordinates, called TopWindow and BottomWindow.
1.1.15. Screen

 - is a grid of character positions, addressed by screen co-ordinates.

1.1.16. Text co-ordinates

 - are pairs of integers defining the line and column position of text characters, with columns counted left to right, 0..255 horizontally and lines counted, 1... downwards, vertically.

1.1.17. Screen co-ordinates

 - are pairs of integers defining the line and column position of screen characters, with columns counted left to right, 0..78 horizontally and lines counted, 0..23 vertically.

1.1.18. Quote line

 - is the bottom (21th) line of the terminal screen.

1.1.19. Cursor position

 - is the separation line between any two characters or between the left margin of the text window and a character.

1.1.20. Cursor addressing

 - is a software capability, that allows the cursor to be moved to a particular screen position.
2. THE COMPUTER INTERFACE

The three dimensions of the computer and user interface of E5863 are

TEXT
SCREEN
and
KEYBOARD

Each of these dimensions are dealt with, in detail, in the following sections, with particular reference to the mechanisms and operators which affect them.

In addition, the computer interface has two global mechanisms, Trace and Error Recovery, that affect all three dimensions, and are therefore dealt with separately.

2.1. TEXT

The ideas upon which the structure of text control is based may be summarised as follows:

* All editing is to be done on a copy of the Input File, called the Edit File. The Input File may be initially empty or non-existent.

* Control characters are removed from the Input File on creation of the Edit File, so they can be used as commands to the editor.

* The Edit File is stored in blocks of 512 bytes with care taken not to split a line between blocks. Thus, each block contains an integer number of text lines.

* Each 512 byte block is stored in the format: <length><text line><length><text line>... where,

  <text line> is a sequence of character codes representing the text.

  <length> is a character which may be interpreted as the number of characters of text in the <text line>. The maximum length is 255 characters (= LINESIZE).

* A linked list of pages is maintained to record the contents of each 512 byte block. Each page records:

  a) a block number, to indicate physical location on the Temporary File used for storage of overflow blocks,
This diagram shows the flow of data through the data structures of the editor, and thus the logical breakdown of the system into subsystems:
b) first line number,

c) link to the next page in the chain, which preserves the sequence of the file indicated by the first line number.

* Before any manipulations can be performed on the text of a block, the block must reside in memory. A number of page frames (at least 3), are maintained in memory, to provide storage for specific blocks of text from the edit file.

* Each frame holds a record of its contents:

a) block number, of the text it contains

b) changed flag, to indicate changed status

c) reference line, to allow implementation of the "Least-Recently-Used" mechanism

d) empty pointer, to the beginning of the unused space in the block

e) buffer (512 bytes), to hold the text.

* The idea is to keep the most recently used blocks in these frames, because it is more likely that the user’s next command will involve text belonging to one of these blocks. This will reduce the disk traffic, thus making the editor quicker and more efficient.

* When reading and manipulating text, operations are performed on specific lines.

Two structures are maintained:

1) the (read-only) "View" Line

2) the "Current" (read-write) Line.

* The (read-only) "View" line is distinguished by pointers to text located in a frame, and is used for applications requiring no changes to be made to the text. Such applications could be screen display, or in searching for a particular phrase.

* The "Current" line is a structure separate to the frames, consisting of a buffer to hold a line of text, and variables to hold its length, line number and changed flag. Any shifting of text (insertion, deletion or replacement), must take place on this line.
2.1.1. Page List Structure and Free List

The Page List structure contains three entities:

1) the block number, which is uniquely allocated by the system to record the location (in 512-byte units) on the temporary file used for storage of overflow blocks. The block numbers are given chronologically, but do not reflect the ordering of the blocks in the Page List. They are used only as an indicator of physical location of blocks on the temporary file.

2) the first line number, which is the number of the first line of text in this block. Line numbers are sequential, starting at 1.

3) a pointer to the next page in the file, which preserves the sequential character of the file.

The page list structure is implemented and maintained as a linked list. The following diagram indicates its nature:

```
      block #0
       |         |
       |         |
       v         v
first line #1
      |         |
      |         |
      v         v
  next

2
   |
   |
   |

3
   |
   |
   |

5
   |
   |
   |

12
  |
  |
  |

nil
```

This linked list of pages records the contents of each 512-byte block of text in the edit file. The first line number gives an indication of which block contains a specific line, and the block number indicates where to (physically) find this block.

To implement such a structure, the following mechanisms are required:
InitPage () -- empty page list

(one empty page)

InsertPage (page, line#) -- block#

(insert a page after <page> with first line number <line#>, where <page> is a pointer to an element of the page list structure)

DeletePage (page) --

(delete the page <page>)

FindPage (page, line#) -- block#

(find the <line#> within the page <page>)

AdjustPage (page, line#, n) --

(add <n> to first line# of all blocks after <page>)

Another list of pages, known as the freelist, is also maintained as a pool of unused pages, to be used as the source, when there is a need to expand the page list of text. This structure removes any upper limit (up to the memory limit of the machine) on the size of files which can be handled by EBS3.

To create the freelist structure, the following mechanism is defined:

MakeFreelist () -- 64 empty pages of freelist

The new pages are given a block number and linked together after allocation.

2.1.2. Frames and Temporary Edit File

A Frame is a structure which contains a 512-byte buffer, which is able to hold an entire block of text. To indicate the contents of this buffer, the following auxiliary variables are stored with the buffer:
1) the block number : is the block number of the text con-
tained in the Frame-buffer.

2) a "changed" flag : a boolean flag used to indicate
whether the text has been changed while residing in the
Frame-buffer.

3) the reference time : which holds a record of when the
frame was last accessed, to allow the implementation of
the "Least-Recently-Used" mechanism.

4) empty pointer : to the beginning of the empty
(unused) space in the block residing in the Frame-
buffer.

Each frame may be pictured as follows:

```
block$
changed
ref. time
empty
```

buffer .... text .... empty space

The Frame structure is defined as an array, with NUM-
FRAMES (at least 3), entries. Working on the assumption
that the user's next command is most likely to involve one
of the blocks which has recently been accessed, the blocks
chosen to occupy the Frames will be those "Most-Recently-
Used".

If the file is too large to fit in the Frame Cache,
then excess blocks will be stored on the temporary file.
Thus, a file of less than NUMFRAMES blocks is stored
entirely in memory.

For any operation to be performed on the text of a
block, the block must occupy a Frame. Thus, a reference to a
line-number which is not contained in a frame causes one of
the frames to be freed (the one Least-Recently-Used), and
the block containing the required line-number is read into
the now empty frame, from the temporary file.

Note that when a frame is freed, if its contents have
been changed while residing in the frame, then it must be
rewritten onto the temporary file into the appropriate posi-
tion. If the contents have not been changed, then it would
be wasteful, resource and time-wise, to rewrite it onto the temporary file.

The following mechanisms need to be provided for the handling of frames:

- **InitFrames ( )** \( \rightarrow \) NUMFRAMES empty, old frames
- **SearchFrames (block\#) \rightarrow frameptr**
  (search frames for a specific block\#)
- **AllocateFrame ( ) \rightarrow frameptr**
  (find "oldest" frame using LRU algorithm)
- **UpdateFrame (frameptr) \rightarrow**
  (reset reference time and changed flag)
- **ClearFrame (block\#) \rightarrow frameptr**
  (clear the frame containing block\#)

For handling the Temporary File:

- **ReadText (frameptr, block\#) \rightarrow** #chars read
- **WriteText (frameptr, block\#) \rightarrow**
  (where text is read into or written from the specified frame (frameptr))

### 2.1.3. The "Current" Line

The "Current" Line structure consists of:

1) a buffer, "CBuffer", of length LINESIZE, which is separate to the frames, and is used to hold a line of text.

2) an integer, "CLength", which denotes the number of characters of text held in CBuffer.
3) an integer, "LineNo", denoting the number of the line held in CBuffer relative to the edit file.

4) a boolean flag, "CChanged", to indicate whether any alterations (insertions, deletions or replacements), have been made to the text in CBuffer during the time it has occupied this space.

Any alteration to a line of text requires that line to reside in CBuffer, so that shifting of text is minimised.

Thus, a line will be brought into CBuffer only when a command is issued to actually change the text of a line.

The position where all insertion, deletion and replacement commands take effect is maintained by another pair of variables:

(Current.Line, Current.Column)

which are text co-ordinates that record the exact location of the cursor in the text.

Note that the line indicated by "Current.Line" will only be brought into CBuffer if an insertion, deletion or replacement command is issued. Hence, there is a distinction between "Current.Line" and "LineNo".

Before a new line is read into CBuffer, the current contents may need to be re-written back to its appropriate position in a Frame (which may require that block to be brought into a Frame). The necessity for this action is indicated by the flag CChanged.

The operators which need to be provided for handling text within the CBuffer are:

InsertChar (c) --> -

(insert <c> at (Current.Line, Current.Column))

ReplaceChar (c) --> -

(replace <c> at (Current.Line, Current.Column))

DeleteChars (n) --> -

(delete n chars starting at (Current.Line, Current.Column))

A number of smaller operators are also required:
BeginWord () → pos

(find the beginning of the word before
 (Current.Line, Current.Column))

EndWord () → pos

(find the end of the word after
 (Current.Line, Current.Column))

EndLine () → pos

(find the end of the line after
 (Current.Line, Current.Column))

2.1. "Read Only Line" : ViewLine

The ViewLine is recognized within the text block of a
frame by a number of pointers and associated variables:

1) VLine, a pointer to the beginning of a line of text

2) VLength, the length of this line

3) VFrame, a pointer to the frame in which the block of
text containing this line resides.

4) VPage, a pointer to the page chain structure, which
determines the location of the block of text within the
edit file.

The ViewLine is used for applications requiring no
change to be made to the text, such as screen displays.

The mechanism to utilize the ViewLine facility is:

ViewLine (VLine) → VPage, VFrame, VLine, VLength

Also required is:

CheckSpace (n) → boolean

(is there enough room in VFrame for n
more bytes?)
If the line does not currently occupy a frame, then it must be retrieved and placed in a frame using Page-chain and Frame mechanisms.

2.1.3 Compound Operators

The operators defined so far have been primitive. To make the system workable, certain compound operators also need to be defined:

SetText (block#) -- frameptr
(retrieve the text indicated by block# and place it into a frame: Tempfile -> Frames)

InsertLine (n) --
(insert the contents of <CBuffer> as line <n> into the edit file (which may require SplitPage, AdjustPage))

DeleteLine (n) --
(delete line <n> from the edit file (ViewLine, AdjustPage, and various Frame operations))

GetLine (n) -- CBuffer, CLength, CLineNo, CChanged
(find line <n> (ViewLine) and place in CBuffer: Pages -> Frames -> CBuffer).

UpdateLine () --
(CBuffer -> Frame )

SplitPage (line#, frameptr) -- new frameptr
(split frame into 2 and move from line# to end, to the new frame. One of these two frames now contains enough room to insert a new line (at least LINESIZE bytes)).

The operations described so far do not necessarily reflect the actual program pieces implemented to perform text manipulation.
2.1.5 Service Routines

CopyStr (from, to, ichars) -

(copies ichars bytes from to)

This is used often in the transfer of
* CBuffer - Frame-buffer
* text within the Frame-buffer
2.2 SCREEN

The Screen dimension of the editor can be divided into three logical sections: screen initialisation, adjustment and termination. Both initialisation and termination require only single mechanisms, while screen adjustment is slightly more complex.

Screen initialisation and termination are largely synonymous with terminal initialisation/termination, and therefore are dependent on the available hardware and its capabilities. In respect to EDB83, the terminal characteristics must be set so that unprintable "control" characters are treated in the same manner as printable characters. A further adjustment needs to be made so that input from the keyboard is not automatically echoed upon the terminal screen. On exit from EDB83, the terminal characteristics must be returned to their original state.

The mechanisms required are:

InitScreen() --> -

(set up terminal characteristics and displays first screen of edit file)

TermScreen() --> -

(restores the original terminal characteristics)

The terminal screen, represented by the text window and the quote line, and the screen cursor, are the "data structures" upon which the screen control mechanisms act. The text window can be accessed in terms of text co-ordinates and screen co-ordinates, but the quote line is independent of the edit file and is therefore defined only by screen co-ordinates.

The position of the screen cursor within the text window can thus be defined in two ways:

1) Current - a text co-ordinate that represents the cursor's position in the edit file, and

2) Cursor - a screen co-ordinate that represents the cursor's position in relation to the screen/text window boundaries.

The major principle in the design of screen control is to keep the "current" portion of the edit file, the portion that contains the cursor, in view in the text window.
Therefore, after each character entry, handled by the keyboard dimension, that causes any change to the text, or movement through the edit file, the screen is immediately updated. In most cases only a single update occurs, to keep the interface between the three dimensions as simple as possible. There are exceptions, where a combination of two updates are required, but these are kept to a minimum.

These updates are handled by the screen adjustment mechanisms. The ideas upon which the structure of screen adjustment is based, may be summarised as follows:

* it must be determined whether the text window has moved. If it has, its position must be updated accordingly. The window should be centred on the cursor's current position. The window must, however, remain within the edit file boundaries.

**NOTE:** that the edit file has no strictly defined upper boundary, as it can (theoretically) be infinitely extended by the user.

* if the text window has been moved, or its contents altered, then the screen display must be updated. The types of changes possible are:

  - one character
  - one line
  - bottom window — current line to the bottom of the window
  - whole window — top to the bottom of the window.

These changes are specified by a screen changes flag, that is set in the keyboard and/or screen dimensions.

* during adjustment of the screen contents, the cursor obviously has to be moved. This is achieved by sending single control characters to the screen, or by cursor addressing. The output operators used for screen output are defined below. When the screen display is complete, the cursor must be moved back to the screen coordinates representing its current position in the edit file.

As the screen is updated after each individual command, this ensures that there is no accumulation of changes to be processed. This results in only three simple screen mechanisms being necessary for screen adjustment.
2.2.1. **Window control**

* to permit relocation of the text window, relative to the text.

MoveWindow() --> -

(redefines the window boundaries so that the window is centred around the current text co-ordinates, where possible)

2.2.2. **Screen update**

* to permit the rewriting of some or all text on the screen.

AdjustScreen() --> -

(adjusts the screen contents, depending on the value of the screen changes flag)

2.2.3. **Cursor control**

* to permit relocation of the physical cursor and update the cursor's screen co-ordinates.

MoveCursor(line, column) --> -

(moves the screen cursor to the screen co-ordinates that correspond to the text co-ordinates (line, column))

An additional operator is necessary to move the cursor to the quote line. The previously defined cursor control mechanism is unsuitable as it is defined in terms of the text, and the quote line is wholly screen dependent.

MoveToQuote --> -

(moves the screen cursor to the first column of the quote line and updates the cursor's screen co-ordinates)
This diagram represents the terminal screen, showing the position of the text window and quote line.

- Screen Cursor
- Screen
- Text Window
2.2.4. Compound Operators.

A set of compound operators, which are utilized by both the screen update and cursor control mechanisms, are the basic input/output functions. These functions handle input from, and output to the terminal screen.

\[ \text{putstring}(s, n) \rightarrow - \]

(writes the character string \( s \), with length \( n \) upon the screen)

\[ \text{putchar}(c) \rightarrow - \]

(writes the character \( c \) upon the screen)

\[ \text{getchar}() \rightarrow c \]

(reads a character \( c \) from the keyboard and returns it)
2.3. KEYBOARD

In designing the interface between the computer and the user, the constraining influence is the point of interface itself, the keyboard. Other forms of interface, such as light pens and printed text scanning are of course available. However, for the sake of portability, and with a concern for producing an editor readily available at the present proposed usage point, the considered interactions have been restricted to those possible with a standard keyboard.

As the editor is driven by the user via input from the keyboard, this section of the design is in essence, the structure which both drives and encloses the rest of the editor. Thus it is the structure within which text control, screen control, tracing and error checking, work and communicate to achieve the desired editing results.

As the keyboard input is the overriding influence for this dimension, the dimension itself is best described simply as KEYBOARD and consists basically of three parts:

1) The overall controlling mechanism for the editor (main-line).

2) Mechanisms to interpret and initiate response to each keyboard input.

3) Mechanisms for performing utility functions, which are required in all three dimensions.

2.3.1. The control mechanism:
* The mainline initialises all global variables that require it. This is either done directly or by other specific mechanisms which act on individual dimensions.

* The name of the file to be edited is obtained. Text control mechanisms are used to create the edit file.

* Screen control mechanisms are used to set up the terminal as required and display the first 22 lines of text in the edit file (if the file being edited is new or empty, a blank screen is displayed).

* The mainline loops, reading and processing each character entered, by summoning other subsidiary KEYBOARD mechanisms. (See the next section)

Response to keyboard entry falls into the following categories:
if the character entered is a printable character, it is entered into the edit file, by insertion or replacement, where appropriate. However, if the user has indicated that a parameter is to be used with the next command, the character is displayed on the quote line and entered into a parameter buffer.

For each non-printable character that is an edit command, the appropriate response is made.

In particular, the entry of the exit command (CTRL-D "04"), followed by an appropriate response to a "save file" request, causes the character entry loop to terminate. Text and Screen termination mechanisms are then summoned, and control is returned to the operating system.

Any erroneous input that is entered, is ignored.

2.3.2. Mechanisms to interpret and initiate response to each keyboard input.

Each of these mechanisms provide a framework within which Text and Screen control achieve the required response.

They each have the following basic format:

i) Set traces as required

ii) If there is a message on the quote line, clear it.

iii) Interpret any parameter passed to it.

iv) If text control operations are needed, call those that are appropriate.

v) Update current line and column numbers if necessary.

vi) Set appropriate flags for screen control so that display can be corrected as required.

vii) If appropriate, place a message on the quote line and indicate that one is there by setting a global flag.

None of the operators which constitute these mechanisms receive parameters. They communicate through global variables.

Only one mechanism, DoExit(), returns a value. This is either a true or false value, and it indicates whether the user has correctly completed the exit sequence. If true, the loop in the control mechanism terminates; otherwise, the
editor continues to process input from the keyboard.


These mechanisms do not directly process input. They are service operators, used in order to simplify the keyboard mechanisms.

CharToInt \( (c) \rightarrow \text{int} \)

(converts a character \(<c>\) to an integer value)

IntToChar \( (i) \rightarrow \text{char} \)

(converts an integer \(<i>\) to a character)

FindLength \( (str) \rightarrow \text{length} \)

(calculates the length of the character string \(<str>\), terminated by \)
3. GLOBAL MECHANISMS

3.1. TRACE

The trace mechanism is a global mechanism which affects all three dimensions. The first executable statement in each procedure of ED263 is of the form

```c
if ( MS_TrOn ) {
    MS_TrLevel = n ;
    Trace("procedure name", MS_TrLevel) ;
}
```

where

- MS_TrOn is a global flag used to turn the trace mechanism ON or OFF
- MS_TrLevel is a global integer variable used to specify the level of trace to be activated

MS_TrOn may be set by a second argument to a.out, so that tracing can be changed without recompilation. If the argument is omitted, tracing is turned OFF by default. If MS_TrOn is set to TRUE, a trace file is created in the current directory. All trace messages are written to this file to prevent interference with the screen display and unnecessary confusion for the user. If the trace file already exists, it is truncated to zero length by the create command.

The trace procedure always prints at least the name of the calling procedure at its lowest level of activation. At higher levels, all the variables that are global to the three dimensions of the editor can be traced. Any local variables that need to be traced are handled separately in the individual procedures of each dimension.

3.2. ERROR HANDLING MECHANISM

ED263 silently ignores erroneous input from the keyboard, by regarding the entries as invalid editing commands. As all edit commands are single key entries, no fatal errors can be caused by keyboard input. Fatal errors can only occur if there are errors in the program logic or code, and would result in program termination. These errors are not recoverable, and should not be, as they signal serious flaws in the editor's software.

The user could also become confused if the screen display is disturbed by external messages, sent by the system or another user. One of the edit commands provided by
EDBB alleviates this problem, by allowing the user to refresh the screen and return to the current position in the edit file.
APPENDIX A

EDBSJ/EDIT DIFFERENCES SUMMARY

* returns unlinked pages from the page list to the free list.

* all entry from the keyboard is processed in the same loop; even characters entered as part of a command parameter, upon the quote line, are processed in this manner.

* an efficient, skip table driven, find algorithm was implemented for both the FIND and BACKFIND commands. (Sublinear Pattern Search: How To Solve It By Computer - R.E. Bryan)

* an overtype mode that functions correctly, was implemented as the editor's default character entry mode.

* the entry of parameters to cursor movement commands has been made consistent. For example: QUOTE J TAB produces a movement of three tabstops in the text file, NOT one tabstop along the Quote Line. (as in EDIT)

* a global replace command was implemented; it is "global" in the file region from the user's current position to the end of the file.

* the following commands were implemented:

  Cut    ~ EDIT's Cut
  MEnd   ~ EDIT's Copy
  Copy   ~
  Repeat

EDBSJ's Cut varies from EDIT's, as the marked position is regarded as a particular text content's position, rather than an absolute line number. Therefore, a user may insert or delete lines of text BEFORE the marked line without affecting his/her image of where the marker has been placed.

* added more informative messages for the user's benefit. In particular, significant cursor movements, through the find, goto or replace commands, and screen redisplay, generate the TELL command message, to inform the user of his/her position in the edit file.

* a limited online Help function (single screen only) has been provided.
* window movement has been kept uniformly to half a window in the required direction, except for explicit full screen movement commands. The window always attempts to centre itself upon the current line or column, depending on the direction of the movement.

* on exit from the editor, the user may make a typing error when deciding whether to save the file or not. We have given the user an opportunity to repair any mistakes. Edh83 accepts the LAST character entered by the user before hitting return, as the user's FINAL decision.
APPENDIX B
ENVIRONMENT IMPROVEMENT

This year for the first time, the Software Project was a group project. Also, the groups were to make use of the Source Code Control System (SCCS), to manage their project code. These two facts made it apparent that in organizing our group's attempt at the project, three issues should be considered in regards to keeping track of code and documentation.

These three were:

1. Simplifying the wide variety and complexity of SCCS calls, so that we had a system that was quick and simple to use.

2. Making sure all code and documentation were easily accessible within each group member's sign on.

3. Easing the problem of having group members unfamiliar with the UNIX file system.

Although the second point automatically implies a breach of standard departmental security policy, it was realized that realistically, at least one member of the group needed access to all logins so that in an emergency, important code and documents could be accessed.

To achieve point (1), a suite of eighteen short shell command files were created, and to satisfy points (2) and (3) a standard .profile file was created and given to each team member. Certain standard directories were also created in each group member's account.

Account Structuring

Each group member had a Project directory created in their accounts and this directory had two two subdirectories: B CSC3, which was to contain all documentation relevant to the project, and ED83, which was to contain all code development for edb83. The public directory assigned to the group, /pub/321/GroupB, was also given two subdirectories with the same names, so that SCCS versions of files could be stored in the same consistent places.

The .profile developed for use by group members, assigned shell variables of two characters to each of the four directories, used in the project and also any other directories unique to a particular account. These variables were all exported so that any directory was directly accessible from any other directory.
Each account had its prompt set up as the owners initials, followed by a $; for example, Jennifer Tran’s prompt is “JTB”. The prompt was not exported, as under the original versions of the software, a standard system prompt of ”$” was the only indication that a user may be in the key (could also have escaped to UNIX from EDIT). Also, having the initials as a prompt made it easy to remember which account you were using, as all of the debugging work was done on one account. Group members wrote their code on their own accounts, before putting it on the public directory, using SCCS.

After setting up shell variables and prompts, the .profile then produced an ls -l listing of the main directory, and a du call to find how much space was being used (an important consideration as the project grew and disk space diminished). A check was then made of the news, and finally mail was read. This was a way of trying to force group members to read their mail, as in a group project, communication is vital, and ignoring mail could have serious consequences.

With the .profile structured in this way, a break, hit as soon as the listing starts, will terminate the .profile but still leave shell variables properly defined. This was useful when debugging, as if the program caused the terminal to hang, you could log in several times in the one session, without having to wade through the information provided.

Handling SCCS

As files were being stored in the public directory assigned to the group, but were being altered and used in the group members’ private directories, even a simple create delta call required a lot of typing. It also required not only a knowledge of SCCS, but one of UNIX file definitions. For example, a create delta call on the file docopy.c would be as follows:

```
delta /pub/JS1/groupB/CDB83/s.docopy.c
```

So in order to simplify SCCS manipulations, and reduce typing, a suite of shell files were developed around six simple manipulations, considered useful on current deltas. When using SCCS the user mostly wants to deal with the latest delta, and in order to reduce confusion within a group project, it was considered best if an attempt to force only manipulation of the latest delta was made.

The suite of files also produced messages informing the user of the calls being made, and performed manipulations on the files obtained, so that EDIT could alter them. The user was also told of any restrictions placed on the use of the file obtained.
The six basic manipulations were

(1) Create and protect a file in SCCS.

(2) Obtain a copy of the file which could not be used for making a delta.

(3) Obtain a copy which could be used for a delta.

(4) Obtain the history of a file.

(5) Make a delta for the file.

(6) Remove a delta.

The shell files created were given obvious names and needed only the file name (no extension), to be passed to them. The files were named as follows:

For .c files  For .d files  For .h files

<table>
<thead>
<tr>
<th>cADMIN</th>
<th>dADMIN</th>
<th>hADMIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>cCOPY</td>
<td>dCOPY</td>
<td>hCOPY</td>
</tr>
<tr>
<td>cGETE</td>
<td>dGETE</td>
<td>hGETE</td>
</tr>
<tr>
<td>cHISTORY</td>
<td>dHISTORY</td>
<td>hHISTORY</td>
</tr>
<tr>
<td>cDETA</td>
<td>dDETA</td>
<td>hDETA</td>
</tr>
<tr>
<td>cRM</td>
<td>dRM</td>
<td>hRM</td>
</tr>
</tbody>
</table>

Only our librarian was given the ADMIN and RM files. It was never found necessary to use the RM files.

Each member of the group had the files to manipulate .h and .c files placed in their /Project/EDBC3 directory and those to handle .d files were placed in their /Project/DOCS3 directory.

A KEY file was also provided for each of these two /Project subdirectories and allowed each member access to the public directory by invoking the Software Key. This KEY file had to be run before invoking the other shell files to manipulate SCCS files.

Thus for a group member to create a new delta of the procedure docopy.c, they need only do the following:

(1) login

(2) cd $PE <RETURN> (This put them in /Project/EDBC3 subdirectory.)

(3) KEY <RETURN>
(4) cGETE docopy <RETURN> ( a gete -e call is performed and if successful the mode of the file obtained is changed so that EDIT can alter its contents. EDIT is then invoked on acquired file. )

(5) Make changes to the file and save it.

(6) cDELTA docopy <RETURN> ( a new delta of the file is created. )

(7) CTRL-d ( Exit from the key. )

(8) CTRL-d ( logs user out. )

Using this system we could easily manipulate our files, find files, and keep our SCCS tree structure simple by forcing work on the latest delta only.

Anything outside the ordinary that had to be done with SCCS ( for example forcing deltas from copies when we had to work on system B over one weekend), were all done by the person that designed the environment so that at all times the whole system appeared consistent and friendly to the less experienced users, and allowed them to accomplish the main task of writing code and documentation without worrying about SCCS.

By using this environment, we at no point ever came up with conflicting versions of files, or suddenly had things stop working, due to the object file being created from outdated versions of the source code. Thus the project development could run smoothly, protected as much as possible from the confusing consequences of SCCS, and the use of combinations of private and public directories. 
APPENDIX C

SYSTEM HIERARCHY CHARTS

The hierarchy charts that follow, are not an exact representation of the procedure and function calls of the source code. They merely give an overview of the interaction between the sub-systems and mechanisms, that combine to form the edb83 screen editor.

Where possible, the mechanisms and operators have been grouped in order of execution, from left to right, along each level of hierarchy.

The charts are ordered as follows:

A. Text Control
B. Screen Control and
C. User Interface.
The mechanisms required to initially create the extra file (remember an input file) and the necessary files and directories.
used to move the cursor to the graphic line

Get object reference

Get window

Get element

Get element

Get element
PART 2

USER GUIDE

SCREEN EDITOR EBB33

TUTORIAL AND USER'S MANUAL

Lynne Thompson
Monika Strasser
Rodney Porter
Jennifer Tran
Dao Trinh

Software Project CSCI321
Group B

University of Wollongong
November 1983
EDB83(i)

NAME
edb83 - screen based text editor

SYNOPSIS
edb83 [ filename ]

DESCRIPTION

edb83 is designed such that the text on the screen always reflects the current contents of the file being edited. The screen acts as a "window" into the text and this text may be altered by simply typing in the required alterations.

These alterations take place at the cursor position, which is taken to be between the character at which the screen cursor (a flashing block) is positioned, and the previous character.

The default editing mode in edb83 is overtype, so text that is typed at the cursor position overwrites the text that is already present.

Alternatively, text may be inserted after the current cursor position by entering CTRL-a (places edb83 in insert mode), followed by the text to be inserted. To return to overtype mode, simply enter CTRL-a again.

More advanced editing operations can be performed as explained in the following pages. These operations are grouped in alphabetical order.

To invoke edb83, type in

    edb83 [ filename ]

Invoking edb83 without a filename argument, will result in a default file edb83.no being edited.

LIMITATIONS

edb83 works on files containing only printable text. On loading the file, any ASCII control characters (other than newline) are stripped out, and tab characters are expanded to the equivalent number of spaces. When saving the file, trailing spaces are removed from each line, and leading spaces converted to tab characters wherever possible.
1. TUTORIAL INTRODUCTION.

1.1. Description.

The most common way to enter text or programs into a
computer is by means of an editor. The Screen Editor
_described in this manual, EDB83, is one of the easiest to
use available for this function.

The EDB83 Screen Editor displays an entire screen of
information during the editing process. A cursor in the
display can be readily moved around the screen to add,
delete or change information.

EDB83 has no "command language" - text is entered by
typing it directly onto the screen, and manipulated by
pressing function keys. Every keystroke has an immediate
visible effect, and the screen display reflects the exact
state of the text at that point in time.

1.2. Getting Started.

The EDB83 Screen Editor makes use of two files: an
INPUT FILE, and an EDIT FILE. The Input File resides on a
system disk, and is the file to be edited. All editing is
done on a copy of the Input File, called the Edit File.

When the editing session is complete, EDB83 asks
whether you want to update the Input File with the Edit
File. A reply of "yes" ('y' or 'Y') will record all the
changes made during the editing session. "No" ('n' or 'N')
will abandon all the changes made, and the Input File will
be left as it was before the editing session.

Of course, when using EDB83 to create a new file, the
Input File is assumed as empty.

You can call EDB83 by typing the following response to
a prompt:

```
$ EDB83 <filename>
```

where <filename>, if an existing file, is the name of
the input file. If <filename> does not already exist, it is
the name of the new file to be created.

To leave the Editor, type CTRL-d.

Now you are ready to begin.
2. USING THE EDB83 SCREEN EDITOR.

2.1. Creating a File.

To start, call the program EDB83. On the same line, enter the name of the file to be created. The EDB83 Editor will respond with its sign-on and a message on the last line of the screen (the quote line).

```
EDITOR EDB83 MAINLINE FILENAME
```

The EDB83 Editor has signed on and informed the user that it could not find a file called <filename>, so it created a NEW FILE with that name. <filename> will be the name of the new input file if a save is requested and no other name is supplied.

You are now ready to enter text into the Edit File by typing printable characters.

2.2. Editing a File.

To retrieve an already existing file, call EDB83 with the name of the file, as was done when creating a file. EDB83 will find the file and treat it as the Input File.

EDB83 signals its readiness to accept commands by displaying the contents of the file (or at least the first 23 lines), and its sign-on and a message will appear on the last line.
This is an example to show how well the BTSS Screen Editor works, and what a wonderful piece of software it is.

You are now ready to insert, delete, change or move through any of the text in the file <filename> by typing printable characters and/or using the function keys.

2.3. The Cursor and Movement Through the File.

The flashing white block which appears on the screen is known as the cursor. All changes to the text occur at the cursor position, which is defined as the position between the cursor and the character to its immediate left.

To move the cursor to any position on the screen, your movement keys are available:

\[ \rightarrow \] will move the cursor to the right by one position.
\[ \leftarrow \] will move the cursor to the left by one position.
\[ \uparrow \] will move the cursor up one line.
\[ \downarrow \] will move the cursor down one line.

To move to the next Tab Stop, use the TAB key.
To backtab to the previous Tab Stop, use the ESC key.

Because the terminal has room to display only 25 lines \* 79 columns of text, it will act as a 'window' into the text. A request to move the cursor beyond the boundaries of the window by using one of the above keys will result in movement of the window across the text by half a screen in the requested direction. Note that this will only occur when the movement is possible, i.e., when the window is not positioned at a boundary of the text.

Larger movements may be made through the file by using
NEXTPAGE (CTRL-n) or PREVPAGE (CTRL-p). NEXTPAGE moves the window downwards by one screen-page (23 lines), and PREVPAGE moves the window upwards by one screen-page. Once again, these actions will not be performed at file boundaries.

2.4. Overtyping and Inserting Text.

The default mode of the EDB83 Screen Editor is over-type. When a printable character is typed, it will appear on the screen at the cursor position, replacing any character which currently exists at this position.

By typing CTRL-o, the mode will switch to INSERT. Now when a printable character is typed, it will be inserted into the text at the cursor position. Characters to the right of the cursor will move right to make room.

Entering the RETURN key will result in a newline being inserted. When typed in the middle of a line, the line will be split. Lines following the cursor are moved downwards. However, when in OVERTYPE mode, the RETURN character results only in a movement of the cursor to the beginning of the next line - no newline is inserted.

To return to OVERTYPE mode, once again type CTRL-o.

2.5. Deleting Text.

Text may be deleted from within a line by positioning the cursor and using one of the following six function keys:

RUBOUT will delete the character to the right of the cursor. Any characters in the line following this position are moved to the left to fill the empty space. At the end of a line, RUBOUT will delete the 'newline', ie, the following line will be joined to the end of the line containing the cursor. All subsequent lines are moved up to fill the empty space.

CLEAR deletes the character to the left of the cursor. Once again, following characters are moved to fill the gap. At the beginning of a line, CLEAR deletes the newline on the previous line, thus joining the current line to the previous, and move subsequent lines up to fill the gap.
will delete the word to the right of the cursor, where a word is defined to be a sequence of letters and/or digits, or a sequence of punctuation marks. If the cursor is within a word when WORDRIGHT is pressed, only those characters in the word to the right of the cursor are deleted.

WORDLEFT

will delete the word to the left of the cursor. If the cursor is within a word when WORDLEFT is pressed, only those characters in the word to the left of the cursor are deleted.

LINERIGHT

will delete all characters from the cursor position to the end of the line (excluding the newline marker).

LINELEFT

will delete all characters from the cursor position to the beginning of the line, shifting any characters following the cursor to the left to close the gap.

Note that WORDRIGHT, WORDLEFT, LINERIGHT and LINELEFT will not delete any 'newline' characters. Either RUBOUT or CLEAR must be used to perform this task.

2.6. The Quote Line.

In order to pass parameters to a function, a line is reserved specifically for the purpose of entering such arguments. For example, to move down by 5 pages, it is less cumbersome to quote "5" and press CTRL-n, rather than to press CTRL-n 5 times and wait for the response 5 times.

By pressing the BREAK key, the cursor is positioned on the quote line, which is always the bottom line on the screen. Any characters subsequently typed will be treated as part of the parameter, until one of the function keys is pressed, or the BREAK key is again pressed. The parameter will now be used by the function requested in order to modify its meaning. The BREAK key will just return the cursor to its original position. After the function has been performed, the cursor will return to its original position in the text, or to a position determined by EDB83 appropriate to the function.

Any of the keys for cursor or page movement and for deleting text have valid meanings when used with a parameter.
2.7. Cut, Copy and Mend.

Facilities are provided to remove or copy portions of text, and to "mend" them into the position marked by the cursor, which may be anywhere in the file. Pressing $HOMF$ invisibly marks one end of the text to be cut or copied. The cursor position at the time of the request will mark the other end.

Press $\text{CTRL-}^{-}\backslash$ to cut text, i.e., to actually remove text from the file and save it in the "mend" buffer.

Press $\text{CTRL-}^{-}c$ to copy text, i.e., don't remove the text from the file, but place it into the "mend" buffer.

The block of text being cut or copied may be within a line, or it may span several lines, or even the whole file. The text is placed into the "mend" buffer, which is elastic, and expands and contracts according to need.

The text held in the "mend" buffer may now be mended into the file at any position, as many times as required by placing the cursor at the required position and pressing $\text{CTRL-}^{-}a$. The contents of the "mend" buffer will remain unchanged until another cut or copy command is issued.

2.8. Repeating Text.

To repeat a block of text a number of times, the repeat command is available. Mark one end of the block to be repeated by $HOMF$ and the other end with the cursor. Press $\text{CTRL-}^{-}r$ to call repeat, and the block of text will be repeated as many times as applicable (due to parameter on the quote line).

Repeat is actually a combination of the copy - mend commands, so the "mend" buffer will contain the block of text to be repeated on completion of this command.

2.9. The Find Facility.

To find a certain string or pattern in the text, type

```
BREAK string $\text{CTRL-}^{-}f$
```

where the string must be in quotes ('string') and may be up to 77 characters in length.

$\text{EDBB3}$ will search forward through the text from the cursor position until it finds the first occurrence of 'string', then it will position the window over the area containing this occurrence. The cursor will be over the first character of the 'string' in the text.
If the 'string' is not found, then no window movement results, but a message will inform the user that no occurrence was found.

The BACKFIND (CTRL-b) performs the same action, except that the search will be made backwards through the file, starting at the cursor position.

If no 'string' is passed to the FIND or BACKFIND functions, then the search will be performed using the 'string' which was last used for a find (ie, it is saved in a "find" buffer). If no 'string' resides in the "find" buffer then no search is performed.

2.10. The Replace Facility.

To replace an occurrence of a 'string' the string must first reside in the "find" buffer - thus a find must first be performed.

Next quote the 'replacement' string and press CTRL-r:

BREAK 'replacement' CTRL-r

Once again the replacement string must be quoted ('replacement').

EDR3 will now replace 'string' with 'replacement'. If no 'replacement' string is quoted, then the contents of the "send" buffer is used - thus large portions of text may be replaced into the text.

To perform a global replace, the parameter on the quote line is quoted as follows:

BREAK 'replacement' all CTRL-r.

All occurrences of the string in the find buffer in the text, from the cursor position to the end of the text will be replaced with 'replacement'.

2.11. Saving the Edit File.

You may save the Edit File at any time during the editing session by typing CTRL-s. If a 'filename' is quoted, then the Edit File will be saved as this Input File.

Note that when leaving the editor, the option is once again given to save the Edit File. At this stage, there is no option to quote a 'filename' - the default is the Input File.
2.12. Other Commands.

EDB83 has other smaller commands which do not disturb the text in any way. A description of these may be found in the reference supplied in the next section. These include commands such as:

- VIEW
- UNIX
- HELP
- GOTO
- TELL

A more detailed description on the use and defaults of all EDB83 facilities is also supplied in the following reference manual.
REFERENCE MANUAL:

BACKFIND

instruction : BackFind

Format : 

[QUOTE] "string" ? CTRL-b

where :

[QUOTE]

is the Quote command key.

string

is the string to be found.

CTRL-b

is the CTRL key.

b

is the b character key.

BackFind is used for finding the first occurrence of a string, previous to the current cursor position.

Steps :

1) Call the QUOTE command.

2) Enter the quoted string to be found.

3) Enter CTRL-b to call the BackFind command: the string will be placed into the find-buffer. While the BackFind is in progress, a message will appear on the Quote line as follows:

FIND PREVIOUS : "string".

If string is found, then the cursor is placed at start of string; otherwise, message will appear as follows:

"string" WAS NOT FOUND.

NOTES :

* If the optional "string" is omitted, then FDB83 will consider the string already in the find-buffer, as the string to be found.

* If the search string contains a single quote ("), then two single quotes (""") must be entered.
eg. "isn't" CTRL-b will find the previous occurrence of the string <isn't>.

CHANGEMODE

instruction : Changemode

format : CTRL-o

    where :

    CTRL-  

    is the CTRL key.

    o  

    is the o character key.

Changemode is used to change the EIBB3 text entry mode from overtype to insert, or insert to overtype.

Steps :

1) Enter CTRL-o to change text entry mode.
CHARLEFT

instruction : CharLeft

Format : CLEAR

where :

CLEAR

is the CLEAR key

CharLeft is used for deleting a single character to the left of the cursor.

Steps :

1) Position the cursor.

2) Enter CLEAR to call the CharLeft command; the character to the left of the cursor will be deleted.

NOTES :

* if the cursor is at the beginning of a line, CharLeft will delete the preceding newline marker (see Glossary), and join the current line to the end of the line above.
CHARRIGHT

instruction : CharRight

Format : RUBOUT

where :

RUBOUT is the RUBOUT key

CharRight is used for deleting a single character to the right of the cursor.

Steps :

1) Position the cursor.

2) Enter RUBOUT to call the CharRight command; the character to the right of the cursor will be deleted.

NOTE :

* if the cursor is at the end of the line, CharRight will delete the newline marker (see Glossary). This will join the following line onto the end of the current line, and any text below it on the screen, will move upwards to close the gap.
COPY

instruction: Copy

format: CTRL-c

where:

CTRL- is the CTRL key

c is the c character key.

Copy is used for copying part of the edit file into the
mend-buffer.

Steps:

1) Set a marker at one end of the portion of text to be
   copied, using the Mark command.

2) Position the cursor at the character following the
   other end of the portion of text to be copied.

3) Enter CTRL-c to call the Copy command: ERB83 will copy
   (without deleting the actual text), the portion of text
   from the position of the marker to the cursor position,
   into the mend-buffer. The message

   Copying requested text.

is displayed. When the task is complete, ERB83 will
inform the user with the message

Copy complete.

NOTES:

* Usually, the Copy command is followed by the Mend com-
  mand, to insert the text into another section of the
  edit file.

* If the marker is in the middle of a line when the text
  is copied, that line will be left justified in the
  mend-buffer.
CUT

instruction : Cut
format : ctrl-\ 
where : 
CTRL- \ is the CTRL key
\ is the \ character key.

Cut is used for deleting a portion of the edit file, and copying it into the send-buffer.

Steps :
1) Set a marker at one end of the portion of text to be cut, using the Mark command.
2) Position the cursor at the character following the other end of the portion of text to be cut.
3) Enter ctrl-\ to call the Cut command; the portion of text marked will be cut out of the text and placed into the send-buffer.

Notes :
* The Cut command may be followed by a Send command, to insert the text into another portion of the edit file.
* if the marker is in the middle of a line when the text is cut, that line will be left justified in the send-buffer.
DOWN

instruction : Down

down

format : "QUOTE n \n LINEFEED"

where :

QUOTE

is the Quote command key.

n

is an optional repeat factor, specifying
the number of lines to move down.

LINEFEED

is the LINEFEED key.

Down is used to move the cursor down one line.

NOTE :

- "QUOTE \n LINEFEED" will move the cursor to the last line
  of the current window, by default.

- "QUOTE n \n LINEFEED", where n < 0, will respond as if
  n=1.
EXIT

instruction : Exit

format : CTRL-d

where :

CTRL-   is the CTRL key.

d       is the d character key.

Exit is used for terminating the editor EDB83, with the option of either updating or not updating the file.

Steps :

1) Enter CTRL-d to call the Exit command. EDB83 will respond :

Text has been changed. Update file (y/n)?

on the Quote line (see Glossary).

2) Enter y (or Y) followed by RETURN, to leave the editor and update the file. EDB83 will respond :

SAVING filename

informing the user that the file is being updated. When the task is complete, EDB83 will respond :

SAVED filename

3) Enter n (or N) followed by RETURN, to leave the editor without updating the file.

NOTES :

* when updating the file, you are allowed to enter any number of characters before hitting RETURN. The last entry is taken as the final decision. Any other character other than y or n, will return you to the editor, as if you had never tried to exit.
FIND

instruction:  Find

format:  [QUOTE "string"] ? CTRL-\f

where:

QUOTE           is the Quote command key
string          is the string to be found.
CTRL-\f          is the CTRL key.
\f               is the \f character key.

Find is used for locating the next occurrence of a string in the edit file, following the current cursor position.

Steps:

1)  Call the QUOTE command.

2)  Enter the quoted string to be found.

3)  Enter CTRL-\f to call the Find command; the string will be placed into the find-buffer. While the Find is in progress, a message will appear on the Quote line as follows:

   FIND NEXT: "string"

If the string is found, then the cursor is placed at start of string, else message will appear as follows:

   "string" WAS NOT FOUND.

NOTES:

*  If the optional "string" is omitted, then PD883 will consider the string already in the find buffer as the string to be found.

*  If the search string contains a single quote ("), then two single quotes (""") must be entered.

eg.  "isn"t" CTRL-\f  will find the next occurrence of the string <isn't>.  
GOTO

instruction : Goto

format : [QUOTE n] CTRL-g

where :

QUOTE is the Quote command key.

n is an optional repeat factor, specifying the line the cursor is to be moved to.

CTRL- is the CTRL key.

g is the g character key.

Goto is used to move the cursor to a specified line or to the position of a marker set by a previous Mark command.

NOTES :

* If the optional line number is omitted, the cursor is moved to the position of the last marker set. If a marker has not been set, this position is the first character in the edit file, by default.
HELP

instruction : Help

format : CTRL-^ 

where :

CTRL-^ is the CTRL character

^ is the ^ (up-hat) character key.

Help is used for displaying a single screen summary of 
the FDB85's editing commands.

Steps :

1) Enter CTRL-^ to call the Help command. A single screen 
display is invoked.

2) To leave the Help screen, enter CTRL-^. This will 
return you to the edit file to continue editing.
LEFT

instruction : Left

format : [ QUOTE n ] <-

where :

QUOTE

is the Quote command key

n

is an optional repeat factor, specifying the number of columns to move left.

<- is the Left Arrow key.

Left is used to move the cursor one column left.

NOTE :

* QUOTE <- will move the cursor to the first column of the current line, by default.

* QUOTE n <-, where n <= 0, will respond as if n=1.
LINELEFT

instruction : lineLeft
format : CTRL-k

where :

CTRL-  is the CTRL key.

k    is the k character key.

lineLeft is used for deleting all characters from the
beginning of the current line to the current cursor posi-
tion.

Steps :

1) Position the cursor.

2) Enter CTRL-k to call the lineLeft command; all charac-
ters from the start of the line to the cursor will be
deleted.

NOTES :

* lineLeft removes characters of a line and puts them
  into the mnd-buffer (see Glossary).

* if the cursor is at the beginning of a line, no charac-
ters are deleted.
LINERIGHT

instruction : Lineright

format : CTRL-1

where :

CTRL-
    is the CTRL key.

1
    is the 1 character key.

Lineright is used for deleting all characters from the current cursor position to the end of the current line.

Steps :

1) Position the cursor.

2) Enter CTRL-1 to call the Lineright command, all characters from the cursor to the end of the line will be deleted.

NOTES :

* Lineright removes characters of a line and puts them into the merr-buffer (see Glossary).

* if the cursor is at the end of a line, no characters are deleted.

* if the cursor is at the start of the line, Lineright will leave a "blank" line on the screen. To delete the newline marker, the Charright command must be used.
MARK

instruction: mark
format: "HOME"

where:

"HOME"

is the HOME key.

Mark will mark the current cursor position.

Steps:

1) Position the cursor.

2) Enter HOME to call the Mark command: EMBR3 accepts the command, but does not give any obvious response.

NOTES:

* If multiple HOME's are entered, the marker will always be reset to the last marked position.
MEND

instruction : Mend

format : CTRL-a

where :

CTRL- is the CTRL key.

a is the a character key.

Mend is used for inserting the contents of the mend-buffer (see Glossary), into the edit file, to the left of the cursor position.

Steps :

1) Position the cursor on the character that the inserted text should precede.

2) Enter CTRL-a: the contents of the mend-buffer will be inserted into the edit file.
NextPage

instruction:   NextPage

format:       [QUOTE n1] [CTRL-n]

where:

QUOTE
    is the Quote command key.

n1
    is an optional repeat factor, specifying
    the number of pages to move forward.

CTRL-n
    is the CTRI key.

n
    is the n character key.

NextPage is used to move the cursor to the next screen
page of text.

NOTES:

* QUOTE CTRI-n moves the cursor to the last page of the
  file, by default.

* QUOTE n1 CTRI-n, where n1 <= 0, will respond as if
  n=1.
PREVPAGE

instruction : PrevPage

format : [ QUOTE n1 ] CTRL-p

where :

QUOTE

is the Quote command key.

n1

is an optional repeat factor, specifying the number of pages to move backward.

CTRL-p

is the CTRL key.

p

is the p character key.

PrevPage is used to move the cursor to the previous screen page of text.

NOTES :

* QUOTE CTRL-p moves the cursor to the first page of the file, by default.

* QUOTE n1 CTRL-p where n1 <= 0, will respond as if n=1.
QUOTE

instruction : Quote

format : BREAK

where :

    BREAK

    is the BREAK key.

Quote is used to enter parameters to be passed to other edit commands.

Steps :

1) Enter BREAK to move the cursor to the Quote Line.

2) Parameters may now be entered on the Quote Line, followed by another edit command.
REFRESH

instruction : Refresh

format : CTRL-v

where :

CTRL- is the CTRL key.

v is the v character key.

Refresh will redisplay the current screen contents.

Steps :

1) Enter CTRL-v to call the Refresh command: the current page is redisplayed, and the Tell command message is also output.
REPEAT

instruction : Repeat

format : [QUOTE n] CTRL-e

where :

QUOTE       is the Quote command key.

n           is an optional repeat factor, specifying the number of times the text block should be repeated.

CTRL-e      is the CTRL key.

e           is the e character key.

Repeat is used to repeat a block of text in the edit file, where a block may be a number of characters, or a number of lines, or a combination of both.

Steps :

1) Set a marker at one end of the block of text to be repeated, using the Mark command.

2) Position the cursor at the character following the other end of the block of text to be repeated.

3) Enter CTRL-e: the block of text will be repeated once.

4) For multiple repeats, the parameter n must be entered before CTRL-e, in the usual manner.

NOTES :

* The block of text is also placed into the send buffer, as it is for the Cut and Copy commands. It may therefore be inserted into the edit file using the Send command, when the repeat is complete.
REPLACE

instruction : Replace
format : [ QUOTE 'string' [CALL] ] CTRL-r

where :
QUOTE
    is the Quote command key.
string
    is the replacement string.
CALL
    is an optional global factor.
CTRL-r
    is the CTRL key.
r
    is the r character key.

Replace is used to replace one, or all occurrences in
the edit file, of the string stored in the find-buffer (see
Glossary), with the replacement string, or the mend-buffer,
if the replacement string is omitted.

Steps :
1) Call the QUOTE command.
2) Enter the quoted replacement string.
3) Enter CTRL-r to call the Replace command: a string that
matches the contents of the find-buffer will be
replaced by the replacement string.
4) If the optional global factor [CALL] is also entered,
all occurrences of the matched string, from the current
position to the end of the file, will be replaced by
the replacement string.

NOTES :
* if no replacement string is passed, then Replace will
use the contents of the mend-buffer as the replacement
string.
* a Find or Backfind command, must be performed before
the Replace command. If the cursor is not positioned
over the first character of the string to be replaced,
the command is ignored.
RETURN

instruction : Return

format : RETURN

where :

RETURN is the RRETURN key.

If the editor is in overtype mode, Return will place the cursor at the beginning of the next line of text.

If the editor is in insert mode, Return will insert a new line and then position the cursor at the beginning of this new line of text.

NOTES :

* QUOTE RRETURN causes a temporary escape to UNIX

* QUOTE string RETURN, executes the <string> as a UNIX command.
RIGHT

instruction : Right

format : [QUOTE n] ->

where :

QUOTE is the Quote command key.

n is an optional repeat factor, specifying the number of columns to move right.

- -> is the Right Arrow key.

Right is used to move the cursor one column right.

NOTE :

* [QUOTE ->] will move the cursor to the column following the last text character on the current line, by default.

* [QUOTE n ->], where n <= 0, will respond as if n=1.
SAVE

instruction : Save

format : [ QUOTE filename ] CTRL-"s"

where :

QUOTE is the Quote command key.

filename is the name of the file, into which the
edit file will be saved.

CTRL-"s" is the CTRL key.

"s" is the s character key.

Save is used for saving the edit file into the original
input file or into a different file.

Steps :

1) call the QUOTE command.

2) Enter the filename into which the text will be saved.

3) Enter CTRL-s: text will be saved to filename and EDB83
will inform the user with the message:

SAVING filename

When the task is complete, EDB83 will inform the user with:

SAVED filename

4) To save the file into the original text file, from
which the edit file was created, simply enter CTRL-s.

NOTES :

# if the user calls the Save command for the original
file, without having changed any of the text, EDB83
will give the message :

FILE : filename IS UPTODATE , NO SAVE NECESSARY.

and the file will not be updated.
TABLEFT

instruction : Tableft

format : [ QUOTE n ] ESC

where :

QUOTE is the Quote command key.

n is an optional repeat factor, specifying the number of tabstops to move left.

ESC is the ESC key.

Tableft is used to move the cursor to the previous tabstop on the current line.

NOTE :

* QUOTE ESC will move the cursor ten tabstops to the left, or to the first column of the current line.

* QUOTE n ESC , where n <= 0, will respond as if n=1.
TABRIGHT

instruction : TabRight

Format : \[QUOTE \ n\ J\ TAB\]

where :

\(QUOTE\)

is the Quote command key.

\(n\)

is an optional repeat factor, specifying the number of tabstops to move right.

\(TAB\)

is the TAB key.

TabRight is used to move the cursor to the next tabstop on the current line.

NOTIE :

* \[QUOTE\ TAB\] will move the cursor ten tabstops to the right, if possible.

* \[QUOTE \ n\ TAB\], where \(n \leq 0\), will respond as if \(n=1\).
TELL

instruction : Tell

format : CTRL-t

where :

CTRL- is the CTRL key.

- is the t character key.

Tell displays an informative message, containing:

1) the name of the editor being used (EDBB3).
2) the name of the file being edited.
3) the current text entry mode (INSERT or OVERTYPE).
4) the line and column position of the cursor.

Steps :

1) Enter CTRL-t to call the Tell command. Information, as described above, will be displayed on the Quote line.
UP

instruction : Up

format : [QUOTE n ] ^

where :

QUOTE is the Quote command key.

n is an optional repeat factor, specifying the number of lines to move up.

^ is the Up Arrow key.

Up is used to move the cursor up one line.

NOTE :

* QUOTE ^ will move the cursor to the first line of the current window, by default.

* QUOTE n ^, where n <= 0, will respond as if n=1.
WORDLEFT

instruction : Wordleft

format : CTRL-q

where :

CTRL- is the CTRL key.

q is the q character key.

Wordleft is used for deleting the word to the left of the cursor.

Steps :

1) Position the cursor.

2) Enter CTRL-q to call the Wordleft command: the word to the left of the cursor will be deleted.

NOTES :

* if the cursor is at the beginning of a line, no characters are deleted.

* if the cursor is within a word, only those characters to the left of the cursor will be deleted.
WORDRIGHT

instruction : WordRight

format : CTRL-w

where :

CTRL-     is the CTRL key.

w

is the w character key.

WordRight is used for deleting the word to the right of the cursor.

Steps :

1) Position the cursor.

2) Enter CTRL-w to call the WordRight command; the word to the right of the cursor will be deleted.

NOTES :

* if the cursor is at the end of the line, no characters are deleted.

* if the cursor is within a word, WordRight will delete only the characters to the right of the cursor.
GLOSSARY

1) command
   A command is an instruction to the computer which specifies an operation to be performed.

2) CTRL-
   A control character is a non-printing ASCII character which is (usually) used to transmit control signals between a peripheral device, and the computer.

3) default
   A reversion to a value already programmed into the FDB editor.

4) definition [3]
   Indicates an option of the elements between the brackets.

5) enter
   Synonymous with "type in".

6) filename
   This is a one to 14 character label which is used to refer to a file.

7) find buffer
   Buffer of 77 chars, containing the last string a Find was requested on.

8) merge buffer
   Buffer which holds characters to be merged into the file. It can contain a few characters, several lines, several pages, up to the length of the file.

9) repeat factor
   Incremental indicator for repeated commands.

10) string
    is a sequence of characters.