ISSN 0105-8525

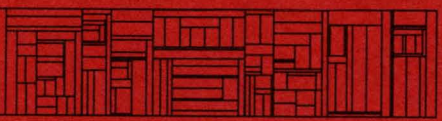
THE RIKKE EDITOR

by

Jens Kristian Kjærgård
Ib Holm Sørensen

DAIMI MD-37
August 1980

Computer Science Department
AARHUS UNIVERSITY
Ny Munkegade – DK 8000 Aarhus C – DENMARK
Telephone: 06 – 12 83 55



ISSN 0105-8525

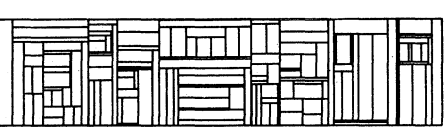
THE RIKKE EDITOR

by

Jens Kristian Kjærgård
Ib Holm Sørensen

DAIMI MD-37
August 1980

Computer Science Department
AARHUS UNIVERSITY
Ny Munkegade – DK 8000 Aarhus C – DENMARK
Telephone: 06 – 12 83 55



1. INTRODUCTION.

The purpose of the editor is to CREATE, UPDATE and MODIFY text files.

The editing commands are divided into groups: those that input and output the contents of the edit buffer and those that modify the edit buffer.

Input commands read a text file (or part of it) into the edit buffer for later editing. The edit commands are issued for the purpose of modifying and changing the contents that were input to the edit buffer by the input commands. When the contents of the edit buffer have been fully updated, the output commands output the corrected text file.

The input file can be modified by insertion, deletion and changing of the original contents by issuing the editing commands. The command structure allows changes to be made on character level as well as on line level. This is accomplished by an implicit character pointer, CP. String searches provide a convenient method of locating characters for examination and/or modification. A macro command facility has been provided to simplify the execution of repetitive command sequences. The editor maintains an implicit line numbering system starting with number 1. This numbering system is continually updated as lines are inserted and deleted.

The command structure of the editor is basically the same as the text editor on the PDP-10, that of TECO.

Input to the editor is a continuous stream of characters. The input is in the form of a text file. This stream of characters is considered to be segmented into pages. A page is a string of characters up to but not including a form feed character. A page is segmented into lines, where a line is defined as a string of characters up to and including a carriage return (CR).

The editor is named edit and is invoked by the command 'edit'.

2. FEATURES OF THE EDITOR.

2.1. ESC (escape) key.

The ESC key is used for two purposes:

1. Striking ESC twice signals the editor to process the command string just typed. A command line is not executed until two consecutive ESC characters are given.

2. Striking ESC once following a string argument to an editor command delimits the argument from the following command sequence or from another string argument.

Striking ESC echoes a "$" on the console.

2.2. Character Pointer (CP).

An implicit character pointer is maintained by the editor and is used to facilitate reaching the exact location for desired modification and examination. This pointer should be thought of as residing between two characters. For example, the CP might be positioned as

```
rikke1
     ↑
     CP
```

(the figure indicates that CP is positioned between "e" and "1".) If characters are inserted, they will be placed at the current position of CP. Insertion of a "-", for example, would give the following string of characters:

```
rikke-1
      ↑
      CP
```

with CP now located at the end of the insertion. Exact directions and commands as how to move and reset the CP are discussed in section 4.

2.3. Rubout.

At any time the user may depress the rubout (RUBOUT) key as a deletion function. This key deletes the last typed character of the command string each time it is depressed. Repeated rubouts delete characters from right to left limited only by deleting the entire command. Should this happen, the editor will issue a new prompt character ("*"). When the rubout key is depressed, it will echo the character deleted on the console.

For example:

```
mary had a baddablittle lamb,
          ↑↑↑
        rubouts
```

will be entered as:

```
mary had a little lamb,
```

2.4. Backspace.

Typing the BACKSPACE key deletes the last typed character in the command buffer and the cursor is moved one step backwards. This may be done repeatedly, also across line bounds.

2.5. Buffer Deletion (CTRLx).

If the user depresses the CTRL key and types "x" during the process of writing a command, the entire command up to that point will be discarded, and the editor will respond with a new prompting "*".

3. COMMAND STRUCTURE.

A command directs the editor to perform a desired operation. Each command consists of a single letter, a double letter or a special graphic, called the command code. The command code can have an optional integer argument n, preceding it, or it can be followed by an optional string argument.

The letter used as command code may only be the lower case letter, except where explicitly stated in the following text. Numeric arguments are always taken as decimal integers.

The editor accepts for execution either a single command or a command string.

3.1. Single Command.

The single command has the following format:

(n)CODE(string$)

where:

n is an optional integer.

CODE is a letter, a double letter or a special graphic.

string represents an optional string of characters.

$ is an echo from depressing the ESC key.

3.2. Command String.

The user can queue two or more commands for sequential execution by the editor. This series of commands is called a command string. Each command code must be delimited from the next command by pressing the ESC key once, which will echo "$" on the console. (If the command cannot be misinterpreted as a parameter to the preceding command, the depression of the ESC key may be omitted. New users are advised not to use this facility, due to the possible catastrophic results from erroneous concatenations of command codes.)

The command string format is

(n)CODE(string$)(n)CODE(string$)...(n)CODE(string$)

Spaces and carriage returns (CR) may be inserted between the command codes to ease reading and to carry the command string over to another line on the console. These extra characters have no effect on the command execution, but are of course significant within a string argument.

3.3. Conventions.

If a control key is depressed (other than ESC, RUBOUT, and CRTL), nothing will be echoed, but the character value is significant. Three characters are ignored on input by the editor:

null	0
line feed	10
delete	127

4. EDITOR COMMANDS.

Editor commands are discussed in the following order in this section:

1. File specification commands.
2. Input commands.
3. Buffer examination commands.
4. CP positioning commands.
5. Edit commands.
6. Output commands.
7. Special commands.

All of the commands are discussed on the pages following and a summary table is provided for reference at the back of this manual.

4.1. File specification commands.

If the editor is invoked with arguments as follows:

```
edit fn ext
```

the editing takes place between the file 'fn ext' and an edit scratch file. Input will be taken from 'fn ext' and output will be written onto the scratch file. Upon exit from the editor (a h command) the files will be closed, the original input file will be copied to the file 'fn BAK' and the scratch file will become the new modified version of file 'fn ext'. The produced output file inherits all attributes from the original input file. If the specified input file does not exist, it will be created as an empty file when entering the editor. Obviously no BAK file will be generated in this case. If only one argument is given at the call of the editor, e.g.

```
edit fn
```

the file 'fn BCPL' will be chosen as input file. Invoking the editor as specified will cause the first page of the input file to be read into the edit buffer. A page is defined as a string of characters terminated by a form feed.

If the editor is invoked with no arguments the user must at least specify the output file. If the user wants to modify an existing

file this file must be specified as an input file. The commands

```
grIfn$Iext$$
gw0fn$0ext$$
```

will select the file 'Ifn Iext' as input file and '0fn 0ext' as output file. The edit buffer will not be affected by these commands.

The command

```
gt$0fn$0ext$$
```

appends output to file '0fn 0ext'.

The two strategies (specifying the input by means of an argument for 'edit' and explicitly using a 'gr' command) must not be confused.

If the user wishes to add new text to a file, the insert command may be used and we refer to its description in section 4.5.

4.2. Input commands.

Read (yank) a page.
By issuing the yank command,

```
y
```

the editor will read a page of the symbolic text from the input file into the edit buffer. The form feed will be read but will not be stored in the edit buffer. The CP will be positioned to the start of the edit buffer. The string will be input until a form feed character is detected or end of file is reached or the edit buffer becomes full. If the capacity of the edit buffer is exceeded, the message:

```
BUFFER IS FULL, Y OR A INPUT TERMINATED
```

is printed on the console. Since this message indicates that a partial page has been read into the edit buffer, the user must output the buffer before the remaining part of the page may be read in.

Append a page.
To append a further page to the present contents of the edit buffer, issue the append command

```
a
```

This will read a page of input from the input file and append it to the end of the text contained in the buffer. The CP will be positioned before the first character of the new page. If, while

appending, the edit buffer's capacity is exceeded, the message:

BUFFER IS FULL, Y OR A INPUT TERMINATED

is printed on the console. The buffer must be output before the rest of the page may be input to the edit buffer.

4.3. Examination of the Buffer.

To type the entire contents of the edit buffer on the console, the user must issue the command:

ht

This command has no effect on the positioning of CP. If the user wishes to examine only a portion of the edit buffer, he should issue the command:

nt

If n is positive n lines are typed from the edit buffer, starting at the current position of CP. If n is negative the preceding n lines are typed, ending just before the current position of CP. n can be omitted in which case one line is typed. CP is not affected by this command.

4.4. Character Pointer (CP) Positioning Commands.

To move CP to the beginning of the edit buffer, the user issues the command

b

Jump to line n.
By issuing the command:

nj

the user can position the CP at the beginning of line n, counting from the start of the edit buffer. The CP will be positioned before the first character of line n.

Move CP n lines.
To move the character pointer from line to line, issue the command:

nl

CP will move n lines from the present position of the CP, and become located before the first character position of the new

line.

for n > 0
: move forward (down) through n carriage returns.

for n < 0
: move backwards (up) through -(n+1) carriage returns and move forward 1 character.

for n = 0
: position CP to the beginning of the current line.

If these operations cause the editor to try to pass the bounds of the text contained in the buffer, the CP is positioned to the first or last character position within the buffer. In this case, the command is equivalent to either the b or the z command.

Move CP to beginning of line.
To move the CP to the beginning of the current line, issue the command:

```
l  (or 0l)
```

CP will move to the beginning of the line, in front of the first character of the line.

Move CP n characters.
To move CP n characters from its present position, the user should issue the command

```
nm
```

where:

for n > 0
: CP is moved n character positions forward (to the right).

for n < 0
: CP is moved n character positions backwards (to the left).

for n = 0
: no effect on CP.

Move CP to end of edit buffer.
By issuing the command:

```
z
```

the CP will be moved to the end of the edit buffer.

The examples on the following pages will illustrate the various possibilities open to the user for moving the CP from character to character and from line to line.

Examples showing the CP positioning commands.
The numbers in parentheses in the following examples represent the implicit line numbering system maintained by the editor.

Let the contents of the edit buffer be:

```
(1)   mary had a little lamb,
(2)   its fleece was white as snow,
(3)   and everywhere that mary went
```

command result

b$$

```
        (1) mary had a little lamb,
            ↑
            CP
        (2) its fleece was white as snow,
        (3) and everywhere that mary went
```

3j$$

```
        (1) mary had a little lamb,
        (2) its fleece was white as snow,
        (3) and everywhere that mary went
            ↑
            CP
```

-1l$$

```
        (1) mary had a little lamb,
        (2) its fleece was white as snow,
            ↑
            CP
        (3) and everywhere that mary went
```

z$$

```
        (1) mary had a little lamb,
        (2) its fleece was white as snow,
        (3) and everywhere that mary went
                                          ↑
                                          CP
```

-10m$$

```
        (1) mary had a little lamb,
        (2) its fleece was white as snow,
        (3) and everywhere that mary went
                                  ↑
                                  CP
```

```
l$$
        (1) mary had a little lamb,
        (2) its fleece was white as snow,
        (3) and everywhere that mary went
            ↑
           CP

45m$$
        (1) mary had a little lamb,
        (2) its fleece was white as snow,
        (3) and everywhere that mary went
                                         ↑
                                        CP

-14l$$
        (1) mary had a little lamb,
            ↑
           CP
        (2) its fleece was white as snow,
        (3) and everywhere that mary went
```

4.5. Text Editing Commands.

Change a string.

A search for a string, deletion of that string and insertion of a new string can be accomplished by issuing the command:

```
cstring1$string2$
```

string1 is searched for in a forward direction from the current CP position. If found, its first occurrence is replaced with string2 , and the CP is positioned to the first character after string2 . If the end of the edit buffer is reached before finding string1 , the message

```
STRING NOT FOUND 'string1'
```

will be printed on the console and CP will be positioned at the beginning of the edit buffer. If the user issues the command

```
cstring1$$
```

the editor simply deletes string1 .

Examples:

1. Contents of edit buffer:
```
        THERE WAS A LITTLEBOY
```
   With the command
```
        cBOY$GIRL$$
```
   the edit buffer is modified to
```
        THERE WAS A LITTLE GIRL
```

2. Contents of edit buffer:
```
        WHO HAD A LITTLY CURLY PIMPLE
```
   After the command
```
        cY PIMPLE$$
```
   the edit buffer will contain
```
        WHO HAD A LITTLE CURL
```

<u>Insert string</u>.
To insert a string of characters, the following command can be issued

```
                istring$
```

string will be inserted at the present position of CP. CP will be repositioned to the end of the inserted string.

The following example shows how insertions are made and where the CP will be positioned. When a line or lines are inserted, subsequent lines are renumbered by the implicit line numbering system maintained by the editor.

Let the edit buffer contain:

```
        WHO HAD A LITTLE CURL
        RIGHT IN THE OF HER FOREHEAD
             ^
             |
            CP
```

Command:
```
        7m$i MIDDLE$0l$1t$$
                      ^  ^
                      |  type one line
        ^  ^
        |  |          move CP to beginning of line

           insert "MIDDLE"

        move CP 7 positions forward
```

After command execution:

```
        RIGHT IN THE MIDDLE OF HER FOREHEAD
       ^
       |
       CP
```

Delete characters.
To delete characters from the edit buffer, the command:

nd

can be used.

for n > 0
delete n characters forward from (to the right of) the present position of CP.

for n < 0
delete n characters backwards from (to the left of) the present position of CP. for n = 0
the command is ignored.

An example:

Before: WHEN SHE WAS GALOOD
CP

Command: -2d$$

After: WHEN SHE WAS GOOD
CP

Delete lines.
To delete lines from the edit buffer, the command

nk

can be issued, where n represents the number of lines to be deleted from the edit buffer.

for n > 0
delete n lines forward from the present position of CP, i.e. delete all characters up to, but not including, the n'th carriage return.

for n < 0
delete -(n+1) lines backwards from the present position of CP.

for n = 0
delete every character before CP's present position back to, but not including, a carriage return.

After execution of the command, the CP will be positioned im-

mediately after the position of the last deleted character. The command can be compared in action with the l command, except that in the case of the k command, each character that CP passes over, is deleted.

The command

```
hk
```

deletes the entire buffer.

For example:

The contents of the edit buffer is:

```
line 1
line 2
 ↑
 CP
line 3
```

command	result
-1k$$	ne 2 line 3
0k$$	line 1 ne 2 line 3
1k$$	line 1 liline 3

Searches.
The editor will perform a search of the edit buffer for a specified string when the command:

```
sstring$
```

is issued. The editor searches forward from the present position of the CP, and is positioned after the last character of the first occurrence of string . If the end of the edit buffer is reached before string is encountered, the message:

```
STRING NOT FOUND 'string'
```

will be printed on the console. If this happens CP will be positioned at the beginning of the edit buffer.

Example:

Before:

```
        THERE WAS A LITTLE GIRL
       ↑
      CP
       WHO HAD A LITTLE CURL
       RIGHT IN THE MIDDLE OF HER FOREHEAD
```

Command:

```
        sOF HER$$

        search for string "OF HER"
```

After:

```
        THERE WAS A LITTLE GIRL
        WHO HAD A LITTLE CURL
        RIGHT IN THE MIDDLE OF HER FOREHEAD
                                   ↑
                                  CP
```

A search for a string may be continued by examining subsequent pages of the input file. This type of search is activated by

nstring$

If string is not found in the current buffer from CP and onwards, the editor outputs the buffer to the output file and brings the next page from the input file and continues the search at the beginning of the new buffer. This process is repeated until the string is found or end of file is encountered.

Should the latter occur, the message

STRING NOT FOUND 'string'

will be printed on the console.

A third way of searching for a string throughout the input stream is provided: It causes examination of subsequent pages for an occurrence of the string but does not output the pages before that page in which string is found. This type of search is called by:

qstring$

If the string is not found in the edit buffer from CP and onwards, the editor yanks a new page into the buffer and continues searching and yanking until the string is found or end of file is reached in which case the message

STRING NOT FOUND 'string'

is written on the console.

In the q and n commands, the CP is positioned immediately after the located string. If the search is unsuccessful, the CP is positioned at the beginning of an empty edit buffer.

Macro command facility.
To ease repetitive command operations, a macro facility has been provided. Only one macro can be defined at a time. To specify the contents of the macro command, the user must write:

```
xmcommand1...commandn$$
```

where command has the format:

```
(n)CODE(string$)
```

(See the earlier description of command structure.) No check for command errors is given during the macro definition. Errors will only be detected during the processing of the macro.

Execution of a macro command is obtained by issuing the command

```
nx
```

where n denotes the number of times the command string of the macro is to be executed.

The commands

```
x$$     0x$$     1x$$
```

are equivalent and execute the macro once.

A defined macro may be deleted at any point in the editing session by issuing the command

```
xd
```

The contents of the macro may be displayed by typing

```
x?
```

For example:

```
b$$
xmcRIKKE1$RIKKE-1$l$1t$$      (macro definition)
x?$$                          (displays macro definition)
1000x$$                       (executes the macro 1000 times)
xd$$                          (deletes the macro)
```

The above sequence of edit commands specifies that all occurrences of "RIKKE1" in the current edit buffer will be changed to "RIKKE-1", assuming "RIKKE1" occurs less than 1000 times.

If an error occurs during execution of the macro, the appropriate

error message will be typed on the console, and the macro execution will be terminated.

If the macro is

1. Undefined
2. Defined recursively (e.g. xm3l$x$$)

the message

MACRO ERROR

will appear on the console writer.

4.6. Output Commands.

Output the buffer.

The following output commands are provided:

p
: Output the entire edit buffer to the output file, and end with a form feed.

np
: Output n lines, starting at the present position of CP, and end with a form feed.

pw
: Output the entire buffer to the output file, no form feed is written at the end.

npw
: Output n lines from the edit buffer, starting at the present position of CP. No form feed is written at the end.

Output buffer and remaining input.

The command

e

will output the current contents of the edit buffer and then copy the remainder of the input file to the output file.

Output buffer and read in new buffer.

The output and read function can be obtained by combining the p and the y commands (e.g. p$y$$). However, the following command accomplishes the same:

r

This operation can be repeated by a

nr

command.

for n < 1
the command is ignored.

for n = 1
the command is equivalent to the r command.

for n > 1
the command will alternately output a page and read in a new page n times.

4.7. Special commands.

command	operation
:	type the number of lines currently in the edit buffer.
.	type the number of the line, that CP is currently pointing to.
=	type the number of characters contained in the edit buffer.
o	Close input, output files and return to the operating system. (must be succeded immediately by ESC)
ex	equivalent to e$o$$

5. SUMMARY.

5.1. Error messages.

Error messages and their meaning are in the table below. They are always printed along with the remainder of the command sequence that was interrupted by the error. Errors always terminate execution of a command sequence.

Error message	Meaning
INPUT TERMINATED	Command string exceeds capacity of the command buffer. Input to the command string is terminated and whatever has been typed in is executed.
BUFFER IS FULL, Y OR A INPUT TERMINATED	The edit buffer's capacity is exceeded. The read is terminated and only a partial page is in the edit buffer.
MACRO ERROR	Macro undefined or called recursively.
COMMAND UNINTELLIGIBLE	Editor cannot parse a command.
STRING NOT FOUND 'string'	A command has failed during a search.

5.2. Summary Table of Editing Commands.

command	format	operation
a	a	Append a page.
b	b	Reset CP to beginning of edit buffer.
c	cstr1$str2$	Search buffer for str1 and change to str2.
d	nd	Delete n characters from position of CP.
e	e	Output buffer and rest of input file.
ex	ex	Equivalent to e$o$$
f	f	Output form feed.
gr	grfnext	Take input from file "fn ext".
gw	gwfnext	Output on file "fn ext".
gt	gtfnext	Append output to file "fn ext".
hk	hk	Delete entire buffer.
ht	ht	Type entire buffer on console.
i	istr$	Insert str at current position of CP.
j	nj	Move CP to beginning of line n.
k	nk	Delete (kill) n lines from the current position of CP.
l	l	Set CP to beginning of present line.
	nl	Set CP at beginning of line, n lines from current position of CP.
m	nm	Move CP n character positions.
n	nstr$	Search for str; if not found, output buffer,read new page and continue search through all input.
o	o	Exit from editor, no output of buffer.
p	p	Output entire buffer, add form feed at end.
	np	Output n lines starting at current position of CP, end with form feed.
pw	pw	Output entire buffer, no form feed at end.
	npw	Output n lines starting at present position of CP, no form feed at end.
q	qstr$	Search for string; if not found, read and continue search through all input.
r	r	Output entire buffer and read in next page.
	nr	Perform r command n times.
s	sstr$	Search edit bufer for str.
t	nt	Type n lines, starting at current position of CP.
x	x	Execute macro once.
	nx	Execute macro n times.
xd	xd	Delete macro definition.
x?	x?	Display current macro definition.
xm	xmstr$	Define str to be a macro command.
y	y	Read (yank) a page.
z	z	Position CP at end of edit buffer.
=	=	Type number of characters in edit buffer.
:	:	Type number of lines in edit buffer.
.	.	Type number of current line.

Kjoerård, Jens Kristian.
The Rikke editor / by Jens Kristian Kjoergård, Ib Holm Sørensen.-- Aarhus, Denmark: Computer Science Department, Aarhus University, 1980.
(DAIMI; MD-37)

I. Joint auth or. II. Title.