**Chapter 1**

**INTRODUCTION**

The interactive text editor has become an important part of almost any computing environment. Editors are the general purpose system software programs, which helps the computer users to enter program or data onto file system of the computer.

In earlier days the concept of file system was not there. Punched cards and tapes were used to enter the program and data. This was a very costly and clumsy technique.The advent of magnetic tapes and disks made the punched card and paper obsolete. The file containing the text, programs or data are kept in secondary storage devices and corrected by the general purpose programs called “EDITORS”.

Example: UNIX’s vi Editor, MS DOS Text Editor, Microsoft’s MS Word, Open offices Word...

The editor may be simple with limited range of operations or may be powerful with sophisticated command.

**1.1 USER INTERFACE**

The different types of the editor are;

1. Line editor
2. Screen editor
3. Graphical editor

Line and screen editors are used to handle the text, where as graphical editors are used to handle figures.

In this project, we are going to implement a *full screen VI EDITOR*

User interface is concerned with the input devices, output devices and interactive language of the system.

**1.1.1 INPUT DEVICES:**

The input devices are classified as

1. Text devices: Type writer like Keyboard(QWERTY Keyboard)
2. Button devices: set of special function keys on an alphanumeric keyboard
3. Locators: mouse and data tablet

The input device we used to implement VI Editor is *QWERTY layout Keyboard.*

**1.1.2 OUTPUT DEVICES:**

The output devices may be

1. Teletype writer or any other character printing terminal.
2. “Glass teletype” based on CRT.
3. Advanced CRT terminals and professional workstations.

The output device we used to implement VI Editor is *Advanced CRT terminal.*

**1.1.3 THE INTERACTION LANGUAGE:**

The interactive language of a text editor can be

1. Typing oriented or text command oriented
2. Menu oriented

The interactive language we used to implement VI Editor is *Typing oriented or text command oriented*

**1.2 VI EDITOR**

**vi** is a family of screen-oriented text editors which share certain characteristics, such as methods of invocation from the operating system command interpreter, and characteristic user interface features. The original vi program as written by *Bill Joy* in 1976 for an early BSD Unix release.

The vi Editor (pronounced vee-eye and it is always written in lower case) is a text editor originally developed for UNIX programmers. It is a modal editor, meaning it operates in three different modes; command mode (default), insert mode, and line mode. The vi Editor has been around for decades and is used daily by thousands of Linux and UNIX users, which is why it is a rite of passage tradition that new Linux and UNIX users learn this tool.

**There are three basic modes of vi:**

**Command mode**

This is the default when you enter vi. In command mode, most letters, or short sequences of letters, that you type will be interpreted as commands, ***without explicitly pressing Enter*** . If you press **Esc** when you're in command mode, your terminal will beep at you.

**Insert mode**

In insert mode, whatever you type is inserted in the file at the cursor position. Type **i** or punch insert key to enter insert mode from command mode; press **Esc** to end insert mode, and return to command mode.

**Line mode**

Use line mode to enter line oriented commands. To enter line mode from command mode, type a colon (: ). Your cursor moves to the bottom of the screen, by a colon prompt. Type a line mode command, and then press **Enter**. Any sensible command from the UNIX line editor will work, and a few are good to know about. These commands are indicated in this handout by a colon in front of the command. Each time you use a line mode command, you must type a colon to enter line mode, then type the command by the colon prompt at the bottom of the screen, then press **Enter** when you finish typing the command

**1.3 NCURSES.H**

Many widely-used programs need to make use of a terminal’s cursor-movement capabilities. A familiar example is the vi (or the vim variation) text editor; most of

its commands make use of such capabilities. For example, hitting the j key while in vi will make the cursor move up one line. Typing dd will result in the current line being erased, the lines below it moving up one line each, and the lines above it remaining unchanged.

The goal of curses library was to alleviate authors of cursor-oriented programs

Like vi of the need to write different code for different terminals. The programs would make calls to the library, and the library would sort out what to do for the given terminal type. For example, if our program needs to clear the screen, it would not (directly) use any character sequences Instead, it would simply make the call clear ();

In order to use curses, you must include in your source code a statement

#include <ncurses.h>

and you must link in the curses library:

cc - sourcefile.c –lncurses

**Chapter - 2**

**System design:**

The VI editor is implemented under Unix platform using C++ language and the curses.h

Library functions. The detailed algorithm is discussed in ALGORITHM section . One of the most important features is it works in command mode and insert mode. The various features and functionalities of the implemented vi editor are:

**READ MODE:**

The functionalities of the various keys are described below:

1. l or left arrow: moves the curser towards left
2. r or right arrow: moves the curser towards right
3. u or up arrow: moves the curser towards up
4. b or down arrow: moves the curser towards down
5. i or insert : enters into input node
6. x or delete: deletes the current character
7. o : inserts the newline c : deletes the line
8. ctrl+l : clears the screen w: returns and saves
9. q: quits the window and doesn't save

**INSERT MODE:**

The functionalities of the various keys are described below:

1. ctrl+d: returns to the read mode
2. ctrl+w: inserts the new line
3. delete: deletes the current character
4. BACKSPACE: deletes previous character
5. up arrow: moves towards up
6. down arrow: moves towards down
7. right arrow: moves towards right
8. left arrow: moves towards left

**LIBRARY FUNCTIONS**

The various curses library functions used to implement Vi editor are given below with details:

**• WINDOW \*initscr():**

REQUIRED. Initializes the whole screen for curses . Returns a pointer to a data structure of type

**WINDOW, used for some other functions.**

• **cbreak():**

Sets the terminal so that it reads characters from keyboard immediately as they are typed, without

waiting for carriage return. Backspace and other control characters (including the carriage return

itself) lose their meaning.

• **nocbreak():**

Restores normal mode.

**• noecho():**

Turns off echoing of the input characters to the screen.

**• echo():**

Restores echo.

**• clear():**

Clears screen, and places cursor in upper-left corner.

**• move(int, int):**

Moves the cursor to the indicated row and column.

* **addch(char):**

Writes the given character at the current cursor position, overwriting what was there before, and

moving the cursor to the right by one position.

**• insch(char):**

Same as addch(), but inserts instead of overwrites; all characters to the right move one space to the

right.

**• mvaddstr(int, int, \*char):**

Moves the cursor to the indicated row and column, and then writes the string to that position.

**• refresh():**

Update the screen to reflect all changes we have requested since the last call to this function. Whatever

changes we make, e.g. by calling addch() above, will NOT appear on the screen until we call

refresh().

**• delch():**

Delete character at the current cursor position, causing all characters to the right moving one space to

the left; cursor position does not change.

**• int getch():**

Reads in one character from the keyboard.

**• char inch():**

Returns the character currently under the cursor.

**• getyx(WINDOW \*, int, int):**

Returns in the two ints the row and column numbers of the current position of the cursor for the given

window.7

**• getmaxyx(WINDOW \*, int, int):**

Returns in the two ints the number of rows and columns for the given window.

**• scanw(), printw():**

Works just like scanf() and printf(), but in a curses environment. Avoid use of scanf() and printf()

in such an environment, which can lead to bizarre results. Note that printw() and scanw() will do

repeated addch() calls, so they will overwrite, not insert.

**• attron(const), attroff()const:**

Turns the given attribute on or off.

**Chapter 3**

**Algorithm**

**2.1 ALGORITHM FOR READ AND DISPLAYING FILE ON STDSCR**

//pool.c is a file which consists of information about how to use the vi editor

//line is a variable which keeps track of how many lines are there in file

//stx and sty denotes current row and column number

//fp is a file pointer

//LINES and COLS are the two built in variables denoting total number of lines and column on stdscr

Step 1: if the number of arguments passed ≠2 then

Display error message and stop

Else

Go to step2

Step 2: open the information file pool.c in read mode

Step 3: display the contents of pool.c

Step 4: input any key

Step 5: close the pool.c file

Step 6: open argv[1] file in append mode //to facilitate opening the new file

Step 7: close the file argv[1]

Step 8: open file argv [1] file in read mode and assign file pointer to fp

Step 9: if fp = NULL then

Display error message and exit

Else

Go to step 10

Step 10: identify the terminal type, perform the first refresh to clear screen and return pointer to stdscr using initscr () function

Step 11: set the current terminal mode to cbreak //cbreak ()

Step 12: perform newline translation using nonl () function

Step 13: disable the echo mode // noecho ()

Step 14: enable insert and deleting the line features // idlok (stdscr,TRUE)

Step 15: enable the keypad translation // keypad(stdscr,TRUE)

Step 16: read the content of file and display it

While (c=getc(fp))!=EOF

if c=’\n’ then

line++

if c>LINES-2

break

addch(c)

Step 17: close the file pointed by fp

Step 18: move the cursor to beginning of the stdscr

Step 19: perform refresh to make affect of addch()

Step 20: edit ()

Step 21: open file argv[1] in write mode and assign pointer to fp

Step 22: write the content present on screen to file

for l=0 to LINES-1 do

n=len(l) //obtain the length of each line

for i=0 to n do

write the character to file

put new line character at EOL

Step 23:close the file pointed by fp

Step 24:unload the screen and return

**2.2 ALGORITHM FOR len() //FINDING THE LENGTH OF EACH LINE**

Step 1: linelen <- COLS-1

Step 2: find the length

While linelen>0 and mvinch(lineno,linelen)=0

Linelen <- linelen-1

Step 3: return linelen+1

**2.3 ALGORITHM FOR edit() //editing the screen**

//c denotes the key stroked

Step 1: while true do

Move(stx,xty)

Perform refresh

Input key and store in c

Switch(c)

Case ‘l’:

Case KEY\_LEFT: if sty>0 sty—

Else flash()

Break

Case ‘r’:

Case KEY\_RIGHT: If sty<COLS-1 then

Sty++

Else flash

Break

Case ‘b’:

Case KEY\_DOWN: If stx<LINES-1 then

Stx++

Else flash

break

Case ‘k’:

Case KEY\_UP: If sty>0 then

Sty--

Else flash

break

Case ‘i’: input()

break

Case KEY\_DC:

Case ‘x’: delch()

break

Case KEY\_IL:

Case o: move(++stx,sty=0)

insertln()

input()

break

Case KEY\_DL:

Case ‘d’: deleteln()

break

Case KEY\_CLEAR:

Case CTRL(‘L’): clear system //system(“clear”)

break

Case ‘w’: return

Case ‘q’: endwin()

exit(2)

default: flash()

**2.4 ALGORITHM FOR input()**

//c denotes the key stroked

Step 1: move string INPUT MODE to bottom line

Mvaddstr(LINES-1,COLS-20,”INPUT MODE”)

Step 2: refresh()

Step 3: while true do

Input key and store it in c

If c=CTRL(‘d’) or c=KEY\_EIC then

Break

Else

If c=CTRL(‘w’) or c=KEY\_ENTER then

Move (++stx,sty=0)

Else

If c=KEY\_DC then

Sty++

Delch()

Else

If c=KEY\_BACKSPACE then

Sty—

Move(stx,sty)

Delch()

Else

If c=KEY\_UP then Stx—

Else

If c=KEY\_DOWN then Stx++

Else

If c=KEY\_LEFT then Sty—

Else

If c=KEY\_RIGHT then Sty++

Else

Inch(c)

Move(stx,++sty)

Perform refresh

**Chapter 6**

**OUTPUT SNAPSHOTS**

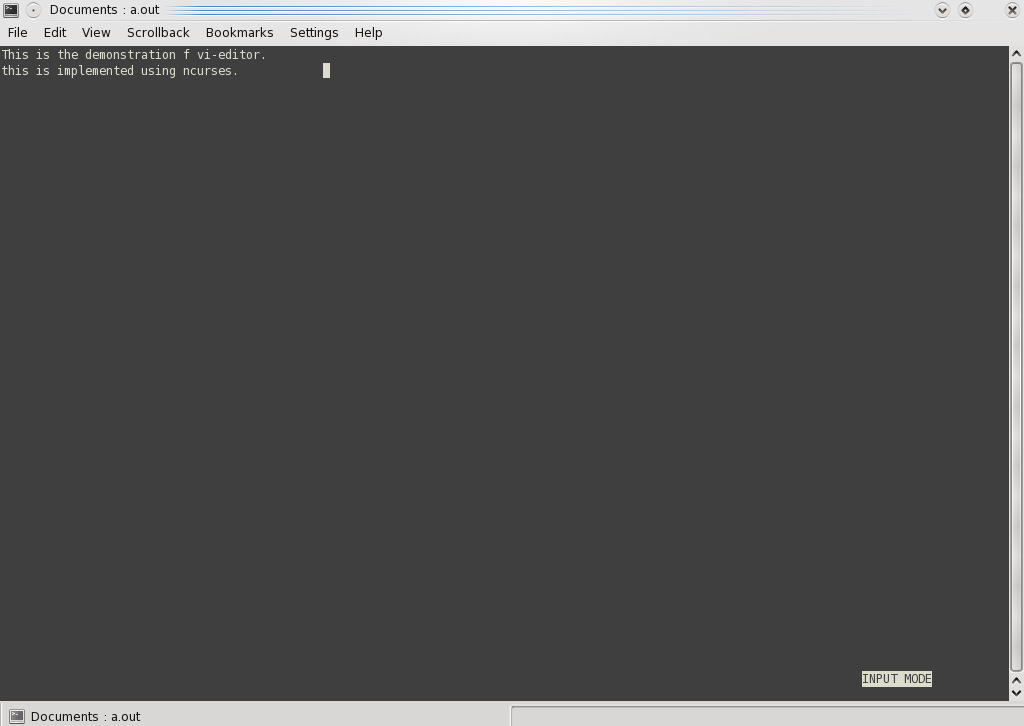


Fig 3.1 INPUT MODE

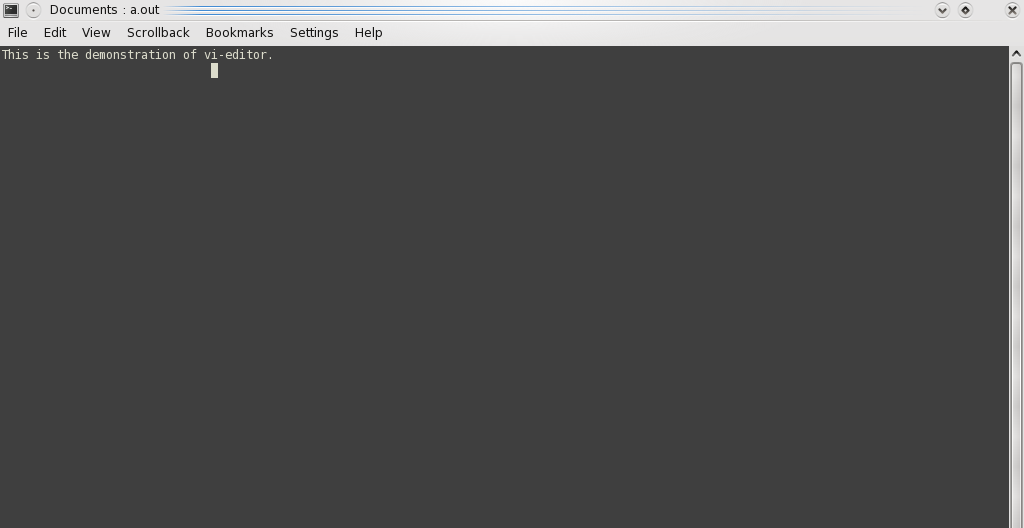


Fig 3.2 DELETE LINE

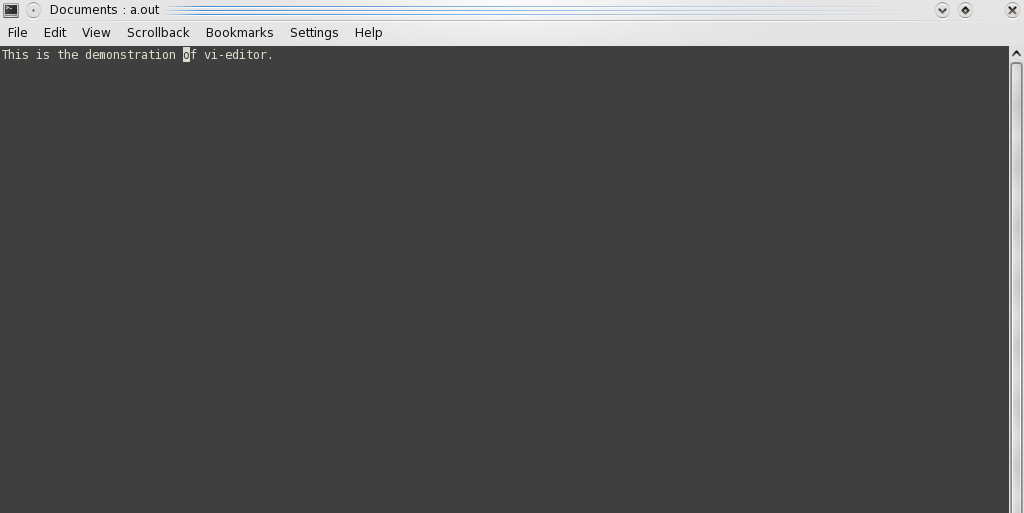


Fig 3.3 (a)POSITIONING THE CURSOR

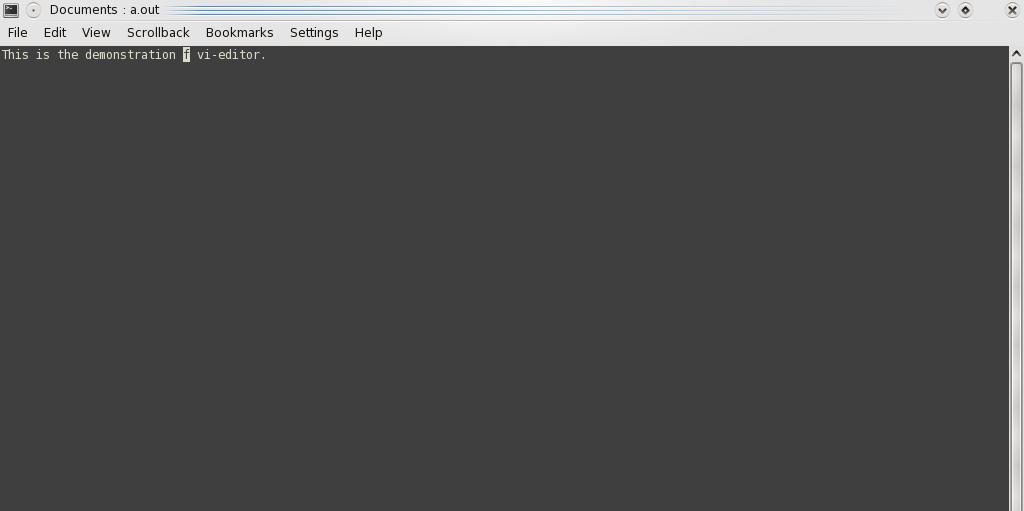


Fig 3.3(b) DELETION OF CHARACTER

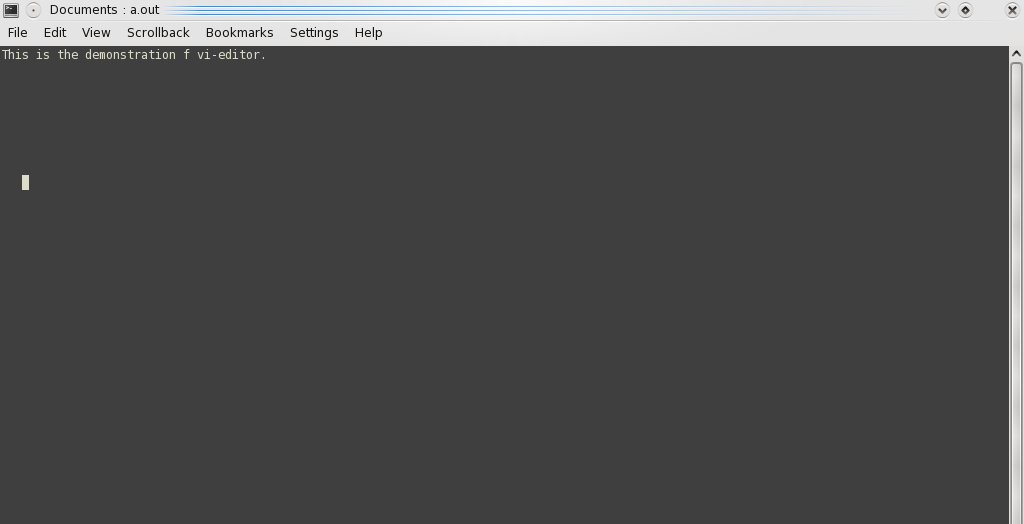


Fig 3.4(a)POSITIONING LEFT

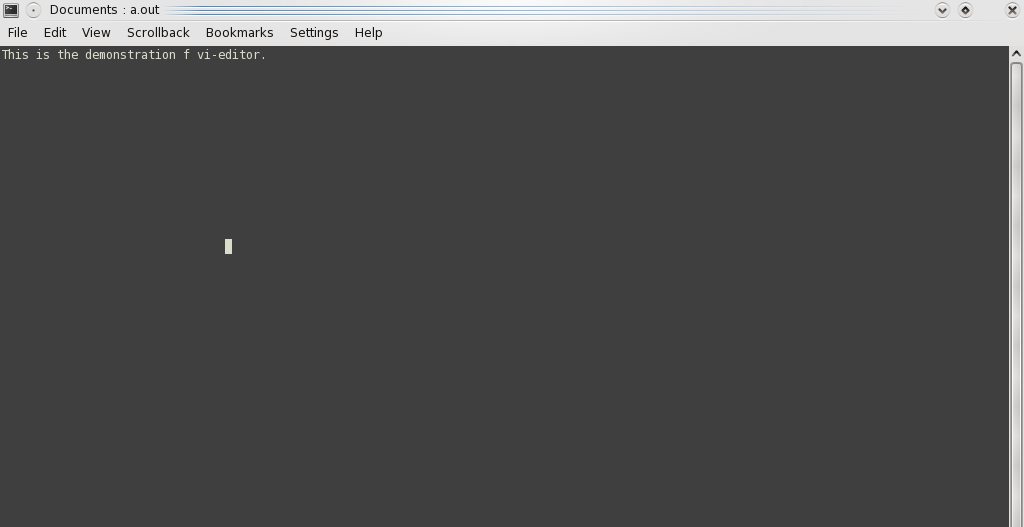


Fig 3.4(b)POSITIONING RIGHT

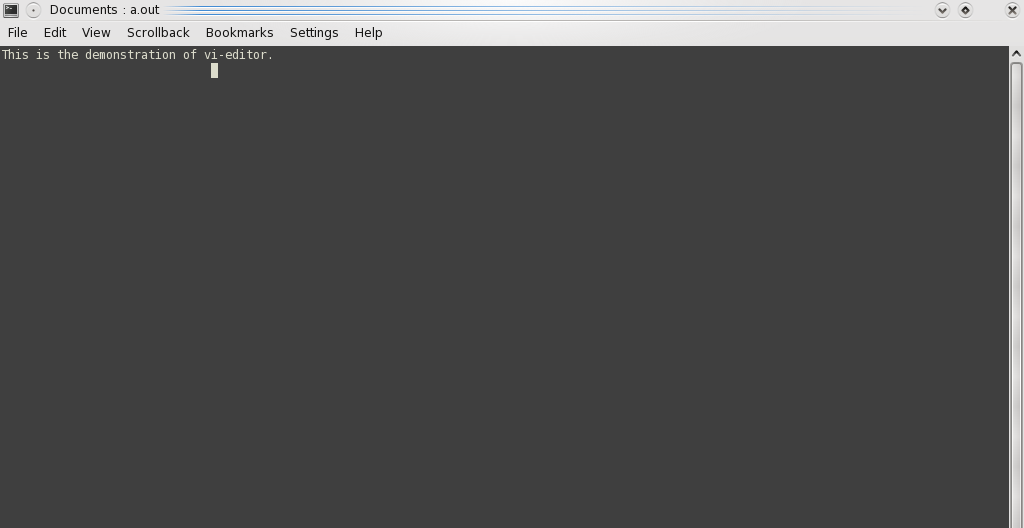


Fig 3.5(a)POSITIONING CURSOR UP

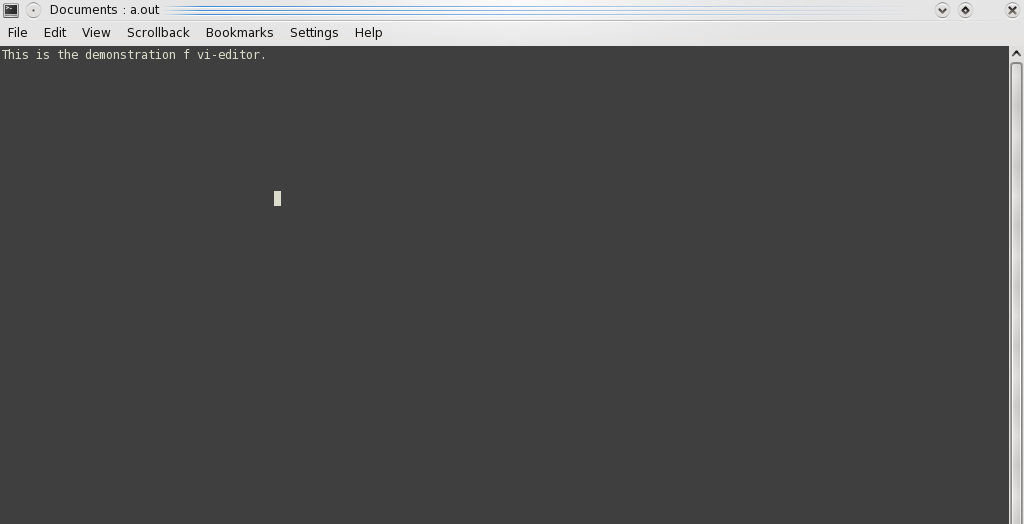


Fig 3.5(b)POSITIONING CURSOR DOWN

**Chapter 5**

**CONCLUSION**

This Vi editor can be efficiently used to open text file for the purpose of reading. Using this software we can also create the new file and the basic file operations like navigation, editing and saving can be done. We hope this project has met its primary goal. This software has been tested under fedora and hope it works in UBUNTU also.

Successful deployment of this project helped us to learn about ncurses library functions. We have come across much recourse which helped us to gain knowledge. We hope this project can be extended very easily.

FUTURE ENHANCEMENT

The various enhancements can be deployed to obtain the much better interface.

* This can be extended to interface with mouse.
* This can be extended to perform find and replace particular text.
* This can be extended to perform search operation.

**Chapter 6**

**BIBLIOGRAPHY**

[1].Norman Matloff,Introduction to the UNIX Curses Library,University of California at Davis 1997-2008,N.Matloff.

[2].The Single Unix Specification,Version 2 1997 The Open Group

[3].http://xnet.fi/susv2/xcurses/curses.h.html