"In Memory Text Editor"

A MINI-PROJECT REPORT

Submitted By:

Kunal Gupta	**ENG17CS0113**
Manoj MR	**ENG17CS0123**
MdZahid Pasha	**ENG17CS0129**
Kushal N	**ENG17CS0115**
Manish Kumar S	**ENG17CS0121**

of

BACHELOR OF TECHNOLOGY

in

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

at

DAYANANDA SAGAR UNIVERSITY

SCHOOL OF ENGINEERING, BANGALORE-560068

DAYANANDA SAGAR UNIVERSITY

School of Engineering, Kudlu Gate, Bangalore-560068

CERTIFICATE

This is to certify that **Mr. KUNAL GUPTA** bearing USN **ENG17CS0113** has satisfactorily completed his/her Mini Project as prescribed by the University for the **III semester B.Tech.** program in Computer Science & Engineering during the year **2018-2019** at the School of Engineering, Dayananda Sagar University, Bangalore.

Date: _____ _____

Signature of the faculty in-charge

Max Marks	Marks Obtained

_____

Signature of Chairman
Department of Computer Science & Engineering

DECLARATION

We hereby declare that the work presented in this mini project entitled -
"In Memory Text Editor", has been carried out by us and it has not been submitted for the award of any degree, diploma or the mini project of any other college or university.

Kunal Gupta (ENG17CS0113)

Manoj MR (ENG17CS0123)

Md Zahid Pasha (ENG17CS0129)

Kushal N (ENG17CS0115)

Manish Kumar S (ENG17CS0121)

ACKNOWLEDGEMENT

We are pleased to acknowledge **Prof. Aishwarya Milan** and **Prof. Srinivasan N** for her invaluable guidance, support, motivation and patience during the course of this mini- project work.

We extend our sincere thanks to our **Chairman Dr. Banga M.K** who continuously helped us throughout the project and without his guidance, this project would have been an uphill task.

We have received a great deal of guidance and co-operation from our friends and we wish to thank one and all that have directly or indirectly helped us in the successful completion of this mini-project work.

Kunal Gupta (ENG17CS0113)

Manoj MR (ENG17CS0123)

Md Zahid Pasha (ENG17CS0129)

Kushal N (ENG17CS0115)

Manish Kumar S (ENG17CS0121)

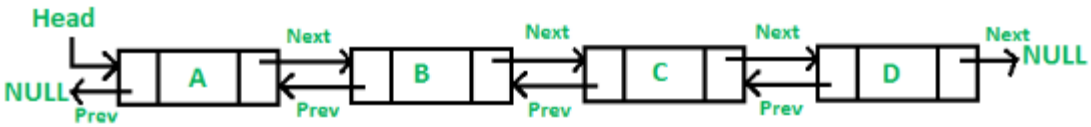
Abstract

The in memory text editor focuses on using Doubly Linked List data structure for storing and accessing text data for using in a text editor. This text editor will be able to directly locate a position on the screen and efficiently able to enter new text data into the doubly linked list.

Contents

Introduction

Doubly Linked List : A **Doubly Linked List** (DLL) contains an extra pointer, typically called previous pointer, together with next pointer and data which are there in singly **linked list**.



Advantages over singly linked list

1) A DLL can be traversed in both forward and backward direction.

2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given.

3) We can quickly insert a new node before a given node.

In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

Disadvantages over singly linked list

1) Every node of DLL Require extra space for an previous pointer. It is possible to implement DLL with single pointer though

2) All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.

TEXT EDITOR:

A ***text editor*** is a tool that allows a user to create and revise *documents* in a computer .It is a computer program that lets a user enter, change, store, and usually print text (characters and numbers, each encoded by the computer and its input and output devices, arranged to have meaning to users or to other programs).

We're using the concept of double linked list to store the data that is entered in the text editor. We have only written an in-memory text editor as a proof of concept for this reason.

About the Problem

To write a text editor that implements a doubly linked list data structure to store the text entered in the GUI.

About the DSA Technique

When the user types a data input key such as an alphabet, number or symbol, a text Node with the corresponding character is added immediately after the node pointed to by the cursor and the cursor is moved one step forward.

We have used the following operations :

- **Insertion** − Adds an element at the current position of the cursor.
- **Display** − Displays the complete list.
- **Delete** − Deletes an element at the current position of the cursor.

1. When the user presses the backspace key the node pointed to by the cursor is removed and the cursor moves one step back.
2. The user should be able to use the direction keys to move the cursor within the text without editing the text. Note: vertical direction keys will move cursor within lines.
3. The user should be able to insert a new line wherever she wishes in text. Note that in this project the newline is taken as a series of spaces till the current line of the window is filled.

Literature Review:

http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/windows.html

S/W & H/W Requirements

Hardware requirements :

Tested on Lenovo ideapad Laptop with components:

- Intel core i7-4700U @ 2.4 GHz
- 12 GB RAM

Software requirements :

- Ubuntu bash terminal.
- 64 Bit Windows/Ubuntu Operating System.

Design:

Pseudo Code:

For inserting node before a new node:

Let the pointer to this given node be next_node and the data of the new node to be added as new_data.

1. Check if the next_node is NULL or not. If it's NULL, return from the function because any new node can not be added before a NULL
2. Allocate memory for the new node, let it be called new_node
3. Set new_node->data = new_data
4. Set the previous pointer of this new_node as the previous node of the next_node, new_node->prev = next_node->prev
5. Set the previous pointer of the next_node as the new_node, next_node->prev = new_node
6. Set the next pointer of this new_node as the next_node, new_node->next = next_node;
7. If the previous node of the new_node is not NULL, then set the next pointer of this previous node as new_node, new_node->prev->next = new_node

For deleting a node in doubly liked list:

Let the node to be deleted is *del*.

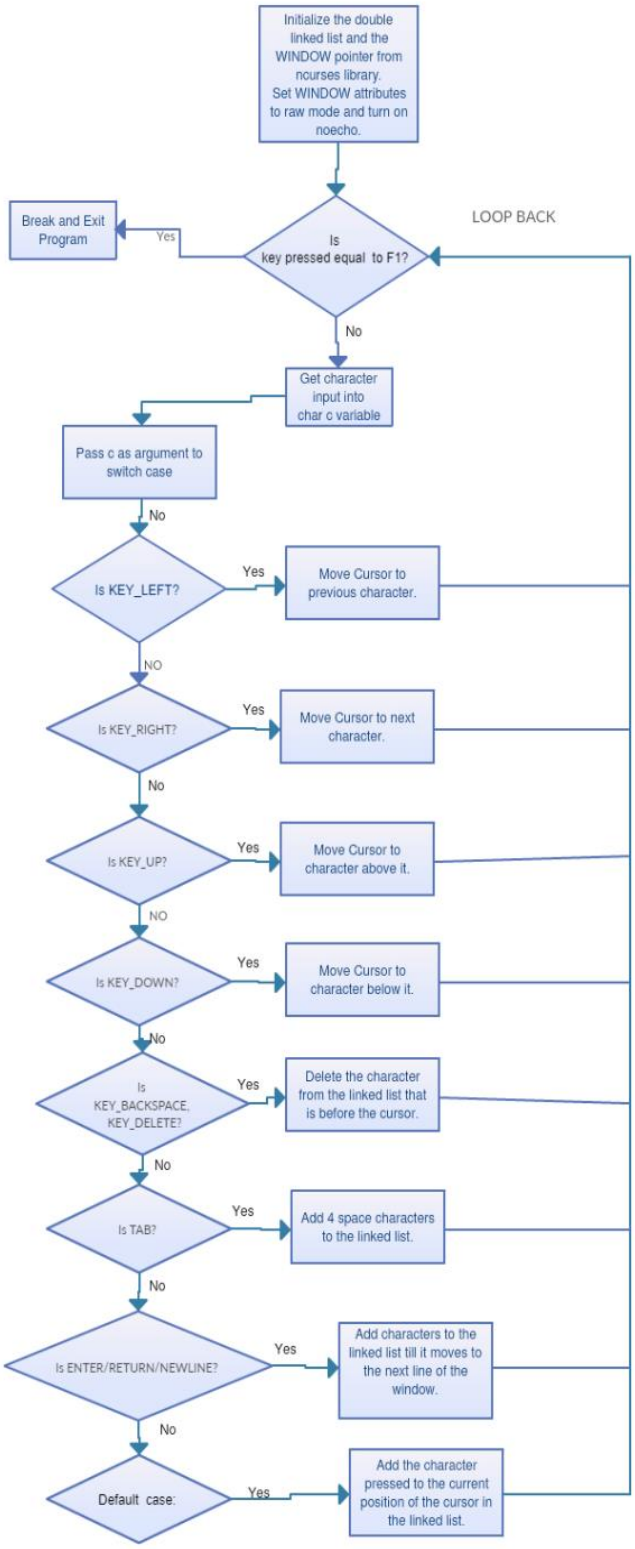
1. If node to be deleted is head node, then change the head pointer to next current head.
2. Set *next* of previous to *del*, if previous to *del* exists.
3. Set *prev* of next to *del*, if next to *del* exists.

For viewing the Doubly Linked List:

Let the current node be Temp

1. Print data at Temp.
2. Assign Temp the address stored in Temp->next_node.
3. Repeat the above steps until Temp->next_node is equal to null.

FlowChart:


Initialize the double linked list and the WINDOW pointer from ncurses library. Set WINDOW attributes to raw mode and turn on noecho.
Is key pressed equal to F1?
Yes
Break and Exit Program
LOOP BACK
No
Get character input into char c variable
Pass c as argument to switch case
No
Is KEY_LEFT?
Yes
Move Cursor to previous character.
NO
Is KEY_RIGHT?
Yes
Move Cursor to next character.
No
Is KEY_UP?
Yes
Move Cursor to character above it.
NO
Is KEY_DOWN?
Yes
Move Cursor to character below it.
No
Is KEY_BACKSPACE, KEY_DELETE?
Yes
Delete the character from the linked list that is before the cursor.
No
Is TAB?
Yes
Add 4 space characters to the linked list.
No
Is ENTER/RETURN/NEWLINE?
Yes
Add characters to the linked list till it moves to the next line of the window.
No
Default case:
Yes
Add the character pressed to the current position of the cursor in the linked list.

CODE:

File Name: text.h

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include<ncurses.h>

#define WIDTH 200
#define HEIGHT 40
#define STARTX 0
#define STARTY 0

#define TAB_WIDTH 4


struct c{
      charch;
      struct c *next, *prev;
};

typedefstruct c Ch;

struct l{
      Ch *first_char, *last_char;
      intno_of_chars;
};

typedefstruct l Line;

Line* convert_str_to_chs(char *str);
Ch* findPos(Line *l, intpos);
Line* insert_line(Line *l, Ch *pos, Line *temp);
Line* insert_at_pos(Line *l, int position, char *str);
Line* delete_at_pos(Line *l, int position);
Line* delete_char(Line *l, Ch* c);
Line* init_line();
void display(Line *l);
voiddisplay_gui(Line *l, WINDOW *win);
intcalc_pos(int y, int x);
```

File Name: text.c

```
#include "text.h"


Line* convert_str_to_chs(char *str)
{
	Line *l = (Line*)malloc(sizeof(Line*));
	l->no_of_chars = strlen(str);
	Ch *temp, *prev;
	for(int i = 0; i < l->no_of_chars; i++)
	{
		temp = (Ch*)malloc(sizeof(Ch*));
		temp->ch = str[i];
		if(i == 0)
		{
			l->first_char = temp;
			prev = temp;
		}
		else
		{
			temp->prev = prev;
			prev->next = temp;
			prev = temp;
		}
	}
	l->last_char = prev;
	return l;
}

Ch* findPos(Line *l, intpos)
{
	Ch *temp = l->first_char;
	int i = 1;
	if(pos == 0)
		return NULL;
	while(i<pos)
	{
		temp=temp->next;
		i++;
	}
	return temp;
}

Line* insert_line(Line *l, Ch *pos, Line *temp)
{
	int chars = 0;
```

```
	chars = temp->no_of_chars;
	l->no_of_chars += chars;

	if(l->first_char == NULL)
	{
		l = temp;
		return l;
	}
	Ch *pos2;

	if(pos != NULL)
		pos2 = pos->next;
	else
		pos2 = l->first_char;

	if(pos != NULL)
		pos->next = temp->first_char;
	else
		l->first_char = temp->first_char;

	(temp->first_char)->prev = pos;

	if(pos2 != NULL)
		pos2->prev = temp->last_char;
	else
		l->last_char = temp->last_char;
	(temp->last_char)->next = pos2;

	return l;
}

Line* insert_at_pos(Line *l, int position, char *str)
{
	if(position > l->no_of_chars+1)
		return l;
	Line *temp;
	temp = convert_str_to_chs(str);
	Ch *pos = findPos(l, position);
	l = insert_line(l, pos, temp);

	return l;
}

Line* delete_at_pos(Line *l, int position)
{
	if(position > l->no_of_chars+1)
		return l;
	Ch *c = findPos(l, position);
```

```
	if(c != NULL)
		delete_char(l,c);
	return l;
}

Line* delete_char(Line *l, Ch* c)
{
	Ch *next = c->next;
	Ch *prev = c->prev;
	if(next != NULL)
		next->prev = prev;
	else
		l->last_char = prev;
	if(prev != NULL)
		prev->next = next;
	else
		l->first_char = next;
	l->no_of_chars -= 1;
	free(c);
	return l;
}

void display(Line *l)
{
	Ch *cur = l->first_char;
	while(cur != NULL)
	{
		//printf("%c", cur->ch); // use for non GUI interface
		printw("%c", cur->ch);
		cur = cur->next;
	}
}

voiddisplay_gui(Line *l, WINDOW *win)
{
	werase(win);
	Ch *cur = l->first_char;
	wmove(win,0,0);
	while(cur != NULL)
	{
		if(cur->ch != '\0')
			wprintw(win, "%c", cur->ch);
		cur = cur->next;
	}
}

Line* init_line()
{
	Line *l = (Line*)malloc(sizeof(Line*));
```

```
	l->no_of_chars = 0;
	l->first_char = NULL;
	l->last_char = NULL;
}

intcalc_pos(int y, int x)
{
	return (WIDTH*y)+x;
}
```

File Name: Main.c

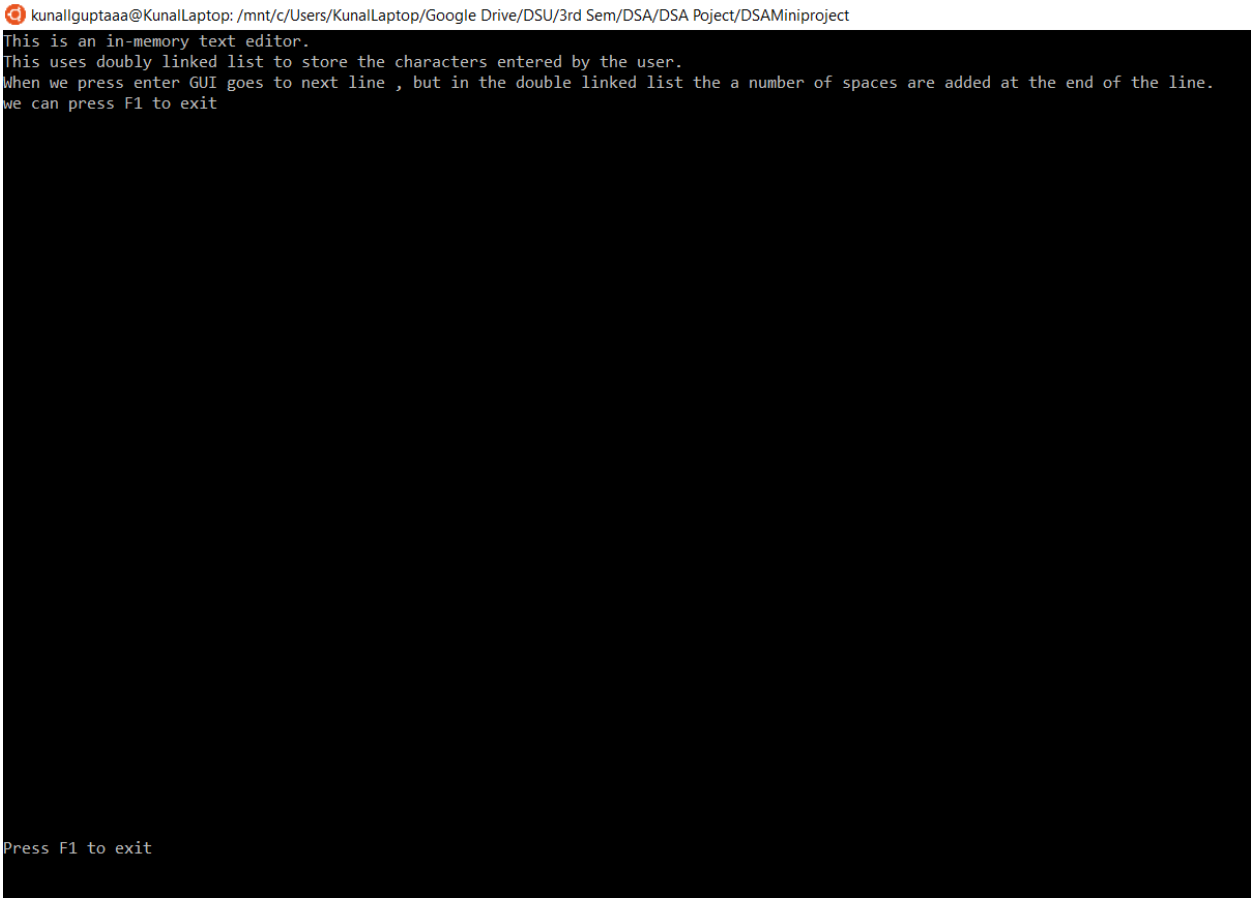
```
#include "text.h"

int main()
{
	WINDOW *win;
	int highlight = 1;
	int choice = 0;
	int c;
	charch[] = " ";

	initscr();
	clear();
	noecho();
	cbreak();
	win = newwin(HEIGHT, WIDTH, STARTY, STARTX);
	keypad(win, TRUE);
	wmove(win, 0, 0);
	//box(win,0,0);
	wrefresh(win);
	Line *l = init_line();
	intpos =0, x = 0, y = 0;
	while(1)
	{
		c = wgetch(win);
		ch[0] = (char) c;
		switch(c)
		{
			case KEY_LEFT:
			{
				if(l->no_of_chars+1 >pos)
				{
					if(x > 0)
						wmove(win,y,--x);
					else if(y!=0)
					{
						x = WIDTH-1;
```

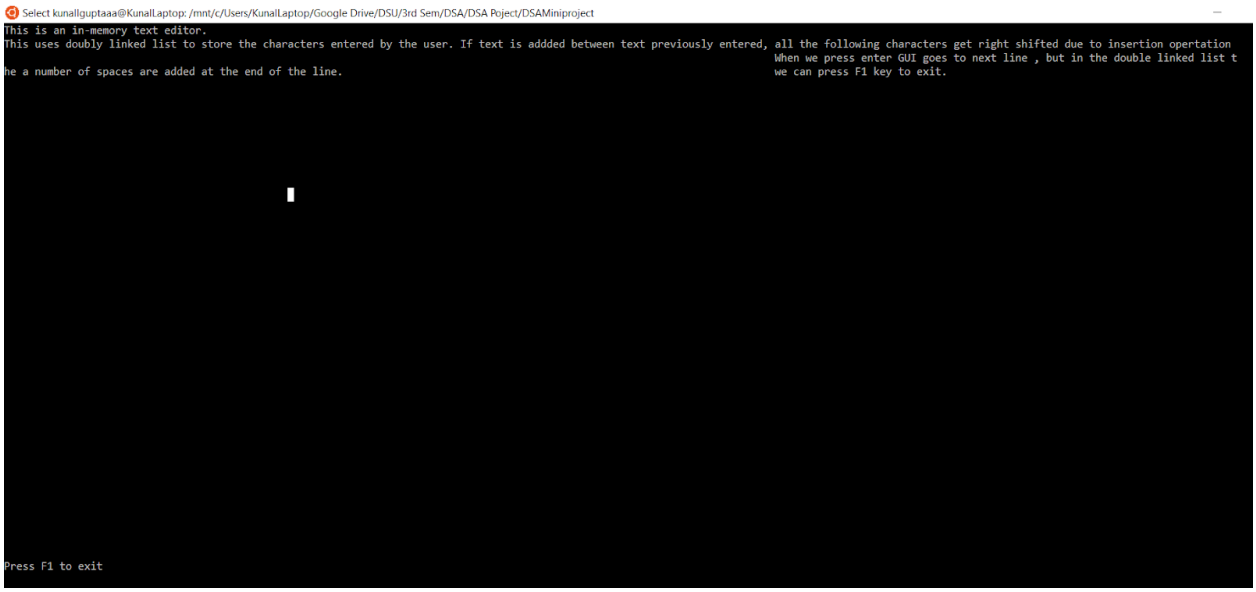
```
                        wmove(win,--y, x);
                }
                pos = calc_pos(y,x);
        }
        break;
}
case KEY_RIGHT:
{
        if(l->no_of_chars+1 > pos+1)
        {
                if(x != WIDTH-1)
                        wmove(win,y,++x);
                else if(y!=HEIGHT-2)
                {
                        x = 0;
                        wmove(win,++y,x);
                }
                pos = calc_pos(y,x);
        }
        break;
}
case KEY_DOWN:
{
        if(l->no_of_chars+1 >pos+WIDTH)
        {
                if(y < HEIGHT-2)
                        wmove(win, ++y, x);
                pos = calc_pos(y,x);
        }
        break;
}
case KEY_UP:
{
        if(l->no_of_chars+1 >pos)
        {
                if(y > 0)
                        wmove(win, --y, x);
                pos = calc_pos(y,x);
        }
        break;
}
case KEY_DC:
case 127:
case KEY_BACKSPACE:
{
        werase(win);
        delete_at_pos(l,pos);
        if(x > 0)
                wmove(win,y,--x);
        else if(y!=0)
```

```
		{
			x = WIDTH-1;
			wmove(win,--y, x);
		}
		pos = calc_pos(y,x);
		display_gui(l,win);
		break;
	}
	case '\t':
	{
		char tab[] = "    ";
		werase(win);
		if(l->no_of_chars+1 >pos)
		{
			l = insert_at_pos(l,pos,tab);
			if(x<WIDTH-1)
				x+=4;
			else if(y<HEIGHT-2)
			{
				x=3;
				y++;
			}
		}
		pos = calc_pos(y,x);
		display_gui(l,win);
		wrefresh(win);
		break;
	}
	case '\n':
	{
		werase(win);
		if(l->no_of_chars+1 >pos)
		{
			for(int i = x; i< WIDTH; i++)
			{
				l = insert_at_pos(l,pos," ");
				if(x<WIDTH-1)
					x++;
				else if(y<HEIGHT-2)
				{
					x=0;
					y++;
				}
			}
		}
		pos = calc_pos(y,x);
		display_gui(l,win);
		wrefresh(win);
		break;
	}
```

```
                case KEY_F(1):
                {
                        clrtoeol();
                        refresh();
                        endwin();
                        exit(0);
                        break;
                }
                default:
                {
                        werase(win);
                        if(l->no_of_chars+1 >pos)
                        {
                                l = insert_at_pos(l,pos,ch);
                                if(x<WIDTH-1)
                                        x++;
                                else if(y<HEIGHT-2)
                                {
                                        x=0;
                                        y++;
                                }
                        }
                        pos = calc_pos(y,x);
                        display_gui(l,win);
                        wrefresh(win);
                }
        }
        mvwprintw(win,HEIGHT-1,0,"Press F1 to exit");
        wmove(win, y, x);
        wrefresh(win);
    }
    clrtoeol();
    refresh();
    endwin();
    return 0;
}
```

Testing and output Screen Shots:





Shortcomings:

If text is added between text previously entered, all the following characters get right shifted due to insertion operation.

Operations required for compilation:

The compilation of the three files needs to be done using this command and the ncurses library is needed to be installed manually.

Execute following command for installing ncurses:

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

Command for compiling the files:

```
gcctext.c -c
gccMain.ctext.o -lncurses -o TextEdit.out
```

Conclusion:

The in memory text editor with the GUI interface is able to store the text entered in the console window and print it back onto the console window in the format it was entered.

References:

[1] Padala, Pradeep. "NCURSES Programming HOWTO."*Tldp.org*, 2005, tldp.org/HOWTO/NCURSES-Programming-HOWTO/index.html.

[2] Weiss, Mark Allen. Data Structures and Algorithm Analysis in C. Dorling Kindersley (India), by Arrangement with Pearson Education, 1997