**TITLE OF THE PROJECT**

TEXT EDITOR

MINI PROJECT REPORT

Submitted in partial fulfillment of the

requirements for the award of the degree of

**BACHELOR OF ENGINEERING**

IN

**INFORMATION TECHNOLOGY**

By

<Preethi konduru>< 1602-19-737-029>

<Vaishnavi Kotha ><1602-19-737-053>



**Department of Information Technology**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Ibrahimbagh, Hyderabad - 31**

**VASAVI COLLEGE OF ENGINEERING(AUTONOMOUS)**

**(AFFILIATED TO OSMANIA UNIVERSITY)**

**HYDERABAD - 500 031**

**Department of Information Technology**

****

**DECLARATION BY CANDIDATE**

We, <Preethi Konduru>**,** <Kotha Vaishnavi>, bearing hall ticket number, **<**1602-19-737-029**>**, <1602-19-737-053>, hereby declare that the project report entitled **<**” Text editor”**>** Department of Information Technology, Vasvi College of Engineering, Hyderabad, is submitted in partial fulfillment of the

requirement for the award of the degree of **Bachelor of**

**Engineering** in **Information Technology**

This is a record of bonafide work carried out by me and the

results embodied in this project report has not been submitted to

any other university or institute for the award of any other degree

or diploma.

**<Preethi konduru>**

**<1602-19-737-029>**

**<Vaishnavi Kotha>**

**<1602-19-737-053>**

**ACKNOWLEDGEMENT**

It gives us immense pleasure to thank the Department of INFORMATION TECHNOLOGY, for introducing the subject “mini-project” in BE third semester, which let us learn and explore more features in “C programming language”.

I would also like to show my appreciation to our honorable principal, Dr. S V RAMANA sir, for supporting us and our beloved mini-project lecturer, Mrs. LEELAVATHI mam, for letting us properly understand the process of doing the mini-project using c and for providing insight and expertise that greatly assisted the project.

My parents were my first teachers and they have provided me with such a great exposure that has helped me bloom. My family and friends will always be loved for sticking by me through thick and thin. THANK YOU!

**Abstract**

When the idea of making a project pitch, the most important step would obviously be writing a code. To write a code, the most important thing would be a text editor which has multiple features and something which makes life easy for programmers while they are trying to write a code. We have come up with the idea of developing a simple text editor using C programming language.

We are also including some of the additional features like highlighting the color of the keywords, search features, and many more useful things. The editor gives us the option of seeing the line count which can help us find where the error is easy if there is one. We can also use arrow keys rather than having to use the mouse all the time. The help bar is going to be a great help if you’re new to using the editor. Syntax highlighting and showing a warning if the code is not saved are other simple yet useful features. The receiver search feature helps to find any word or line of the code and also makes accessing lines easier.

**INTRODUCTION**

Our project is “ text editor”. Basically, a text editor is a type of computer program that edits plain text. Such programs are sometimes known as notepad. Some text editors are also used to change file configurations. This can be used to write code in different programming languages, such as c, c++, java, ruby, etc..,

The text editor that we have built is a basic text editor that contains all features that a minimal text editor has and also with syntax highlighting and search features. This project is made in C language and in the Linux environment.

Many programmers are often found in various text editors. Some editors like vim, which is user friendly. Our text editor is similar to vim. It is capable of writing codes in C and C++ programming languages. The main aim of our project is to implement a basic text editor, where a user can edit the plain text as well as use it as a C and C++ text editor.

Building this project involved many C libraries as well as methods in a specific order.

* Setting the environment.
* Changing terminal to raw mode.
* Taking raw input to output
* A text editor
* Adding search feature
* Syntax highlighting.

By following the above steps we finally made a text editor with which we can open a plain text file or a C file or a C++ file. A new file can also be created along with the extensions of the file like “.c”, “.cpp”, etc..,

Text editors have a feature set different from that of traditional word processing programs. For example, most won’t let you include pictures or include tables, or double space your writing. The features of the text editor vary from implementation to implementation. But there are several kinds of features that most editors have.

Some of the features that are included in our program of a text editor are saving a file, searching characters in the file, syntax highlighting for keywords while writing code, this really helps a lot because this text editor is mainly used by programmers to code in C and C++.

**TECHNOLOGY**

To implement any project successfully, there will be technological requirements which can either be software or hardware requirements.

**a) Software requirements**

Since our project was supposed to be based on the C programming language, it is a bare necessity to have the knowledge and syntaxes of the language and a proper compiler and a text editor to run and write the programs. Our project also includes some of the C libraries which are not supported by the windows or mac operating system. They can be run in a Linux environment.

1

Linux environment: Some of the windows subsystems for Linux are

1. Ubuntu
2. Kali Linux
3. OpenSUSE - Leap-15-1
4. Debian
5. Alpine WSL

These are the subsystems that provide a Linux environment for our windows operating system. We chose Debian as our subsystem for this project.

Compiler

Many of the C compilers include:

1. Borland Turbo C
2. Tiny C compiler
3. Portable C compiler
4. GCC compiler
5. Clang

Among the many available compilers, we have installed and used the GCC compiler to run/execute the code for “Image Steganography” that we have written.

Text editor

To actually write and complete a code in any language, a text editor is important. Some of the famous text editors are:

1. Vim editor
2. Notepad
3. Notepad++

From the above text editors, we chose vim text editor to write our code and execute it properly as it is user friendly in a Linux environment.

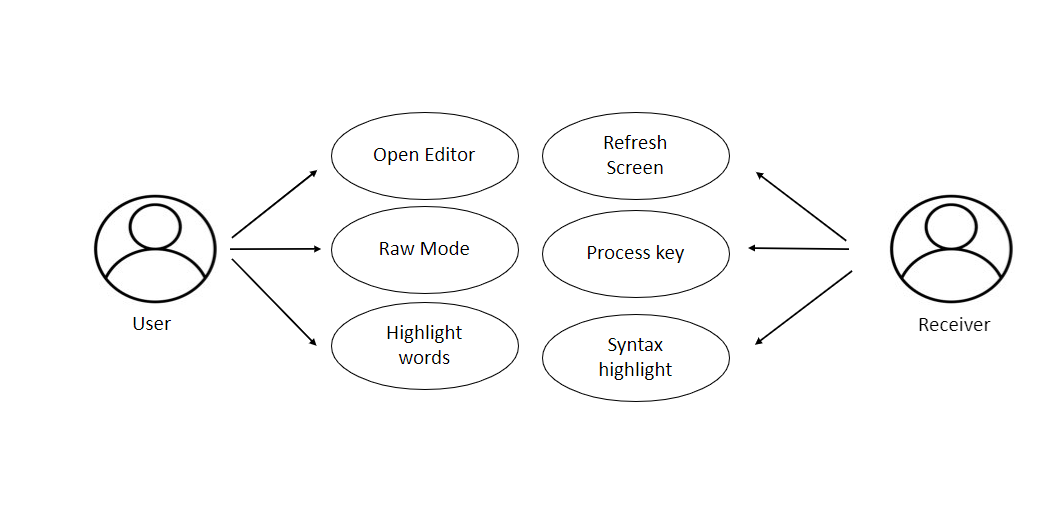
**b) Hardware requirements**

1. Personal Computer or a Laptop with a Linux subsystem.

**Proposed work**

**a) Design**

**I. Use case Diagram:**



**II. Flow**

The main method ‘Open Editor’ function is called then it allows the user to open the editor, then it initializes all the values to 0. Updates the window size and the editor opens for the user.

When the user chooses raw mode then, the echo is turned off and then the computer turns into the canonical mode. When the user enters the input it stores the data into termios structure. On exiting the mode it disables the raw mode.

Words will be highlighted.

Then the screen gets refreshed and programs get executed and first it prints the number of lines and name of the saved file. In the end, the receiver can save files by pressing ctrl-s, can find the word by pressing ctrl-f if the word is matched then the word will be highlighted.

**b.Implementation**

Below are the modules in the program for the text editor.

#define KILO\_VERSION "0.0.1"

#define KILO\_TAB\_STOP 8

#define KILO\_QUIT\_TIMES 3

#define CTRL\_KEY(k) ((k)&0x1f)

#define HL\_HIGHLIGHT\_NUMBERS (1 << 0)

#define HL\_HIGHLIGHT\_STRINGS (1 << 1)

enum editorKey {

BACKSPACE = 127, ARROW\_LEFT = 1000, ARROW\_RIGHT, ARROW\_UP, ARROW\_DOWN, DEL\_KEY, HOME\_KEY, END\_KEY, PAGE\_UP,

PAGE\_DOWN,

};

enum editorHighlight {

HL\_NORMAL = 0, HL\_COMMENT, HL\_MLCOMMENT, HL\_KEYWORD1,

HL\_KEYWORD2, HL\_STRING, HL\_NUMBER, HL\_MATCH

};

struct editorSyntax {

char \*filetype; char \*\*filematch; char \*\*keywords; char \*singleline\_comment\_start;

char \*multiline\_comment\_start; char \*multiline\_comment\_end; int flags;

};

typedef struct erow {

int idx; int size; int rsize; char \*chars; char \*render; unsigned char \*hl;

int hl\_open\_comment;

} erow;

struct editorConfig {

int cx, cy; int rx; int rowoff; int coloff; int screenrows; int screencols; int numrows;

erow \*row; int dirty; char \*filename; char statusmsg[80];

time\_t statusmsg\_time; struct editorSyntax \*syntax;

struct termios orig\_termios;

};

struct editorConfig E;

char \*C\_HL\_extensions[] = {".c", ".h", ".cpp", NULL};

char \*C\_HL\_keywords[] = {"switch", "if", "while", "for", "break",

"continue", "return", "else", "struct", "union",

"typedef", "static", "enum", "class", "case",

"int|", "long|", "double|", "float|", "char|",

"unsigned|", "signed|", "void|", NULL};

struct editorSyntax HLDB[] = {

{"c", C\_HL\_extensions, C\_HL\_keywords, "//", "/\*", "\*/",

HL\_HIGHLIGHT\_NUMBERS | HL\_HIGHLIGHT\_STRINGS},

};

void die(const char \*s) {

write(STDOUT\_FILENO, "\x1b[2J", 4);

write(STDOUT\_FILENO, "\x1b[H", 3);

perror(s);

exit(1);

}

void disableRawMode() {

if (tcsetattr(STDIN\_FILENO, TCSAFLUSH, &E.orig\_termios) == -1)

die("tcsetattr");

}

void enableRawMode() {

if (tcgetattr(STDIN\_FILENO, &E.orig\_termios) == -1)

die("tcgetattr");

atexit(disableRawMode);

struct termios raw = E.orig\_termios;

raw.c\_iflag &= ~(BRKINT | ICRNL | INPCK | ISTRIP | IXON);

raw.c\_oflag &= ~(OPOST);

raw.c\_cflag &= ~(CS8);

raw.c\_lflag &= ~(ECHO | ICANON | IEXTEN | ISIG);

raw.c\_cc[VMIN] = 0;

raw.c\_cc[VTIME] = 1;

if (tcsetattr(STDIN\_FILENO, TCSAFLUSH, &raw) == -1)

die("tcsetattr");

}

int editorReadKey() {

int nread;

char c;

while ((nread = read(STDIN\_FILENO, &c, 1)) != 1) {

if (nread == -1 && errno != EAGAIN)

die("read");

}

if (c == '\x1b') {

char seq[3];

if (read(STDIN\_FILENO, &seq[0], 1) != 1)

return '\x1b';

if (read(STDIN\_FILENO, &seq[1], 1) != 1)

return '\x1b';

if (seq[0] == '[') {

if (seq[1] >= '0' && seq[1] <= '9') {

if (read(STDIN\_FILENO, &seq[2], 1) != 1)

return '\x1b';

if (seq[2] == '~') {

switch (seq[1]) {

case '1':

return HOME\_KEY;

case '3':

return DEL\_KEY;

case '4':

return END\_KEY;

case '5':

return PAGE\_UP;

case '6':

return PAGE\_DOWN;

case '7':

return HOME\_KEY;

case '8':

return END\_KEY;

}

}

} else {

switch (seq[1]) {

case 'A':

return ARROW\_UP;

case 'B':

return ARROW\_DOWN;

case 'C':

return ARROW\_RIGHT;

case 'D':

return ARROW\_LEFT;

case 'H':

return HOME\_KEY;

case 'F':

return END\_KEY;

}

}

} else if (seq[0] == '0') {

switch (seq[1]) {

case 'H':

return HOME\_KEY;

case 'F':

return END\_KEY;

}

}

return '\x1b';

} else {

return c;

}

}

int getCursorPosition(int \*rows, int \*cols) {

char buf[32];

unsigned int i = 0;

if (write(STDOUT\_FILENO, "\x1b[6n", 4) != 4)

return -1;

while (i < sizeof(buf) - 1) {

if (read(STDIN\_FILENO, &buf[i], 1) != 1)

break;

if (buf[i] == 'R')

break;

i++;

}

buf[i] = '\0';

if (buf[0] != '\x1b' || buf[1] != '[')

return -1;

if (sscanf(&buf[2], "%d;%d", rows, cols) != 2)

return -1;

return 0;

}

int getWindowSize(int \*rows, int \*cols) {

struct winsize ws;

if (ioctl(STDOUT\_FILENO, TIOCGWINSZ, &ws) == -1 || ws.ws\_col == 0) {

if (write(STDOUT\_FILENO, "\x1b[999C\x1b[999B", 12) != 12)

return -1;

return getCursorPosition(rows, cols);

} else {

\*cols = ws.ws\_col;

\*rows = ws.ws\_row;

return 0;

}

}

int is\_separator(int c) {

return isspace(c) || c == '\0' || strchr(",.()+-/\*=~%<>[];", c) != NULL;

}

void editorUpdateSyntax(erow \*row) {

row->hl = realloc(row->hl, row->rsize);

memset(row->hl, HL\_NORMAL, row->rsize);

if (E.syntax == NULL)

return;

char \*\*keywords = E.syntax->keywords;

char \*scs = E.syntax->singleline\_comment\_start;

char \*mcs = E.syntax->multiline\_comment\_start;

char \*mce = E.syntax->multiline\_comment\_end;

int scs\_len = scs ? strlen(scs) : 0;

int mcs\_len = mcs ? strlen(mcs) : 0;

int mce\_len = mce ? strlen(mce) : 0;

int prev\_sep = 1;

int in\_string = 0;

int in\_comment = (row->idx > 0 && E.row[row->idx - 1].hl\_open\_comment);

int i = 0;

while (i < row->rsize) {

char c = row->render[i];

unsigned char prev\_hl = (i > 0) ? row->hl[i - 1] : HL\_NORMAL;

if (scs\_len && !in\_string && !in\_comment) {

if (!strncmp(&row->render[i], scs, scs\_len)) {

memset(&row->hl[i], HL\_COMMENT, row->rsize - i);

break;

}

}

if (mcs\_len && mce\_len && !in\_string) {

if (in\_comment) {

row->hl[i] = HL\_MLCOMMENT;

if (!strncmp(&row->render[i], mce, mce\_len)) {

memset(&row->hl[i], HL\_MLCOMMENT, mce\_len);

i += mce\_len;

in\_comment = 0;

prev\_sep = 1;

continue;

} else {

i++;

continue;

}

} else if (!strncmp(&row->render[i], mcs, mcs\_len)) {

memset(&row->hl[i], HL\_MLCOMMENT, mcs\_len);

i += mcs\_len;

in\_comment = 1;

continue;

}

}

if (E.syntax->flags & HL\_HIGHLIGHT\_STRINGS) {

if (in\_string) {

row->hl[i] = HL\_STRING;

if (c == '\\' && i + 1 < row->rsize) {

row->hl[i + 1] = HL\_STRING;

i += 2;

continue;

}

if (c == in\_string)

in\_string = 0;

i++;

prev\_sep = 1;

continue;

} else {

if (c == '"' || c == '\'') {

in\_string = c;

row->hl[i] = HL\_STRING;

i++;

continue;

}

}

}

if (E.syntax->flags & HL\_HIGHLIGHT\_NUMBERS) {

if ((isdigit(c) && (prev\_sep || prev\_hl == HL\_NUMBER)) ||

(c == '.' && prev\_hl == HL\_NUMBER)) {

row->hl[i] = HL\_NUMBER;

i++;

prev\_sep = 0;

continue;

}

}

if (prev\_sep) {

int j;

for (j = 0; keywords[j]; j++) {

int klen = strlen(keywords[j]);

int kw2 = keywords[j][klen - 1] == "|";

if (kw2)

klen--;

if (!strncmp(&row->render[i], keywords[j], klen) &&

is\_separator(row->render[i + klen])) {

memset(&row->hl[i], kw2 ? HL\_KEYWORD2 : HL\_KEYWORD1, klen);

i += klen;

break;

}

}

if (keywords[j] != NULL) {

prev\_sep = 0;

continue;

}

}

prev\_sep = is\_separator(c);

i++;

}

int changed = (row->hl\_open\_comment != in\_comment);

row->hl\_open\_comment = in\_comment;

if (changed && row->idx + 1 < E.numrows)

editorUpdateSyntax(&E.row[row->idx + 1]);

}

int editorSyntaxToColor(int hl) {

switch (hl) {

case HL\_COMMENT:

case HL\_MLCOMMENT:

return 36;

case HL\_KEYWORD1:

return 33;

case HL\_KEYWORD2:

return 32;

case HL\_STRING:

return 35;

case HL\_NUMBER:

return 31;

case HL\_MATCH:

return 34;

default:

return 37;

}

}

void editorSelectSyntaxHighlight() {

E.syntax = NULL;

if (E.filename == NULL)

return;

char \*ext = strrchr(E.filename, '.');

for (unsigned int j = 0; j < HLDB\_ENTRIES; j++) {

struct editorSyntax \*s = &HLDB[j];

unsigned int i = 0;

while (s->filematch[i]) {

int is\_ext = (s->filematch[i][0] == '.');

if ((is\_ext && ext && !strcmp(ext, s->filematch[i])) ||

(!is\_ext && strstr(E.filename, s->filematch[i]))) {

E.syntax = s;

int filerow;

for (filerow = 0; filerow < E.numrows; filerow++) {

editorUpdateSyntax(&E.row[filerow]);

}

return;

}

i++;

}

}

}

int editorRowCxToRx(erow \*row, int cx) {

int rx = 0;

int j;

for (j = 0; j < cx; j++) {

if (row->chars[j] == '\t') {

rx += (KILO\_TAB\_STOP - 1) - (rx % KILO\_TAB\_STOP);

}

rx++;

}

return rx;

}

int editorRowRxToCx(erow \*row, int rx) {

int cur\_rx = 0;

int cx;

for (cx = 0; cx < row->size; cx++) {

if (row->chars[cx] == '\t')

cur\_rx += (KILO\_TAB\_STOP - 1) - (cur\_rx % KILO\_TAB\_STOP);

cur\_rx++;

if (cur\_rx > rx)

return cx;

}

return cx;

}

void editorUpdateRow(erow \*row) {

int tabs = 0;

int j;

for (j = 0; j < row->size; j++) {

if (row->chars[j] == '\t')

tabs++;

}

free(row->render);

row->render = malloc(row->size + tabs \* (KILO\_TAB\_STOP - 1) + 1);

int idx = 0;

for (j = 0; j < row->size; j++) {

if (row->chars[j] == '\t') {

row->render[idx++] = ' ';

while (idx % KILO\_TAB\_STOP != 0) {

row->render[idx++] = ' ';

}

} else {

row->render[idx++] = row->chars[j];

}

}

row->render[idx] = '\0';

row->rsize = idx;

editorUpdateSyntax(row);

}

void editorInsertRow(int at, char \*s, size\_t len) {

if (at < 0 || at > E.numrows)

return;

E.row = realloc(E.row, sizeof(erow) \* (E.numrows + 1));

memmove(&E.row[at + 1], &E.row[at], sizeof(erow) \* (E.numrows - at));

for (int j = at + 1; j <= E.numrows; j++)

E.row[j].idx++;

E.row[at].idx = at;

E.row[at].size = len;

E.row[at].chars = malloc(len + 1);

memcpy(E.row[at].chars, s, len);

E.row[at].chars[len] = '\0';

E.row[at].rsize = 0;

E.row[at].render = NULL;

E.row[at].hl = NULL;

E.row[at].hl\_open\_comment = 0;

editorUpdateRow(&E.row[at]);

E.numrows++;

E.dirty++;

}

void editorFreeRow(erow \*row) {

free(row->render);

free(row->chars);

free(row->hl);

}

void editorDelRow(int at) {

if (at < 0 || at >= E.numrows)

return;

editorFreeRow(&E.row[at]);

memmove(&E.row[at], &E.row[at + 1], sizeof(erow) \* (E.numrows - at - 1));

for (int j = at; j < E.numrows - 1; j++)

E.row[j].idx--;

E.numrows--;

E.dirty++;

}

void editorRowInsertChar(erow \*row, int at, int c) {

if (at < 0 || at > row->size)

at = row->size;

row->chars = realloc(row->chars, row->size + 2);

memmove(&row->chars[at + 1], &row->chars[at], row->size - at + 1);

row->size++;

row->chars[at] = c;

editorUpdateRow(row);

E.dirty++;

}

void editorRowAppendString(erow \*row, char \*s, size\_t len) {

row->chars = realloc(row->chars, row->size + len + 1);

memcpy(&row->chars[row->size], s, len);

row->size += len;

row->chars[row->size] = '\0';

editorUpdateRow(row);

E.dirty++;

}

void editorRowDelChar(erow \*row, int at) {

if (at < 0 || at >= row->size)

return;

memmove(&row->chars[at], &row->chars[at + 1], row->size - at);

row->size--;

editorUpdateRow(row);

E.dirty++;

}

void editorInsertChar(int c) {

if (E.cy == E.numrows) {

editorInsertRow(E.numrows, "", 0);

}

editorRowInsertChar(&E.row[E.cy], E.cx, c);

E.cx++;

}

void editorInsertNewLine() {

if (E.cx == 0) {

editorInsertRow(E.cy, "", 0);

} else {

erow \*row = &E.row[E.cy];

editorInsertRow(E.cy + 1, &row->chars[E.cx], row->size - E.cx);

row = &E.row[E.cy];

row->size = E.cx;

row->chars[row->size] = '\0';

editorUpdateRow(row);

}

E.cy++;

E.cx = 0;

}

void editorDelChar() {

if (E.cy == E.numrows)

return;

if (E.cx == 0 && E.cy == 0)

return;

erow \*row = &E.row[E.cy];

if (E.cx > 0) {

editorRowDelChar(row, E.cx - 1);

E.cx--;

} else {

E.cx = E.row[E.cy - 1].size;

editorRowAppendString(&E.row[E.cy - 1], row->chars, row->size);

editorDelRow(E.cy);

E.cy--;

}

}

/\*\* file i/o \*\*/

char \*editorRowsToString(int \*buflen) {

int totlen = 0;

int j;

for (j = 0; j < E.numrows; j++)

totlen += E.row[j].size + 1;

\*buflen = totlen;

char \*buf = malloc(totlen);

char \*p = buf;

for (j = 0; j < E.numrows; j++) {

memcpy(p, E.row[j].chars, E.row[j].size);

p += E.row[j].size;

\*p = '\n';

p++;

}

return buf;

}

void editorOpen(char \*filename) {

free(E.filename);

E.filename = strdup(filename);

editorSelectSyntaxHighlight();

FILE \*fp = fopen(filename, "r");

if (!fp)

die("fopen");

char \*line = NULL;

size\_t linecap = 0;

ssize\_t linelen;

while ((linelen = getline(&line, &linecap, fp)) != -1) {

while (linelen > 0 &&

(line[linelen - 1] == '\n' || line[linelen - 1] == '\r'))

linelen--;

editorInsertRow(E.numrows, line, linelen);

}

free(line);

fclose(fp);

E.dirty = 0;

}

void editorSave() {

if (E.filename == NULL) {

E.filename = editorPrompt("Save as: %s", NULL);

if (E.filename == NULL) {

editorSetStatusMessage("Save aborted");

return;

}

editorSelectSyntaxHighlight();

}

int len;

char \*buf = editorRowsToString(&len);

int fd = open(E.filename, O\_RDWR | O\_CREAT, 0644);

if (fd != -1) {

if (ftruncate(fd, len) != -1) {

if (write(fd, buf, len) == len) {

close(fd);

free(buf);

E.dirty = 0;

editorSetStatusMessage("%d bytes written to disk", len);

return;

}

}

close(fd);

}

free(buf);

editorSetStatusMessage("Can't save! I/O error: %s", strerror(errno));

}

void editorFindCallback(char \*query, int key) {

static int last\_match = -1;

static int direction = 1;

static int saved\_hl\_line;

static char \*saved\_hl = NULL;

if (saved\_hl) {

memcpy(E.row[saved\_hl\_line].hl, saved\_hl, E.row[saved\_hl\_line].rsize);

free(saved\_hl);

saved\_hl = NULL;

}

if (key == '\r' || key == '\x1b') {

last\_match = -1;

direction = 1;

return;

} else if (key == ARROW\_RIGHT || key == ARROW\_DOWN) {

direction = 1;

} else if (key == ARROW\_LEFT || key == ARROW\_UP) {

direction = -1;

} else {

last\_match = -1;

direction = 1;

}

if (last\_match == -1)

direction = 1;

int current = last\_match;

int i;

for (i = 0; i < E.numrows; i++) {

current += direction;

if (current == -1)

current = E.numrows - 1;

else if (current == E.numrows)

current = 0;

erow \*row = &E.row[current];

char \*match = strstr(row->render, query);

if (match) {

last\_match = current;

E.cy = current;

E.cx = editorRowRxToCx(row, match - row->render);

E.rowoff = E.numrows;

saved\_hl\_line = current;

saved\_hl = malloc(row->size);

memcpy(saved\_hl, row->hl, row->rsize);

memset(&row->hl[match - row->render], HL\_MATCH, strlen(query));

break;

}

}

}

void editorFind() {

int saved\_cx = E.cx;

int saved\_cy = E.cy;

int saved\_coloff = E.coloff;

int saved\_rowoff = E.rowoff;

char \*query =

editorPrompt("Search: %s (Use ESC/Arrows/Enter)", editorFindCallback);

if (query) {

free(query);

} else {

E.cx = saved\_cx;

E.cy = saved\_cy;

E.coloff = saved\_coloff;

E.rowoff = saved\_rowoff;

}

}

struct abuf {

char \*b;

int len;

};

#define ABUF\_INIT \

{ NULL, 0 }

void abAppend(struct abuf \*ab, const char \*s, int len) {

char \*new = realloc(ab->b, ab->len + len);

if (new == NULL)

return;

memcpy(&new[ab->len], s, len);

ab->b = new;

ab->len += len;

}

void abFree(struct abuf \*ab) { free(ab->b); }

/\*\*\* output \*\*\*/

void editorScroll() {

E.rx = 0;

if (E.cy < E.numrows) {

E.rx = editorRowCxToRx(&E.row[E.cy], E.cx);

}

if (E.cy < E.rowoff) {

E.rowoff = E.cy;

}

if (E.cy >= E.rowoff + E.screenrows) {

E.rowoff = E.cy - E.screenrows + 1;

}

if (E.rx < E.coloff) {

E.coloff = E.rx;

}

if (E.rx >= E.coloff + E.screencols) {

E.coloff = E.rx - E.screencols + 1;

}

}

void editorDrawRows(struct abuf \*ab) {

int y;

for (y = 0; y < E.screenrows; y++) {

int filerow = y + E.rowoff;

if (filerow >= E.numrows) {

if (E.numrows == 0 && y == E.screenrows / 3) {

char welcome[80];

int welcomelen = snprintf(welcome, sizeof(welcome),

" ---Basic text editor--- ");

if (welcomelen > E.screencols)

welcomelen = E.screencols;

int padding = (E.screencols - welcomelen) / 2;

if (padding) {

abAppend(ab, "~", 1);

padding--;

}

while (padding--)

abAppend(ab, " ", 1);

abAppend(ab, welcome, welcomelen);

} else {

abAppend(ab, "~", 1);

}

} else {

int len = E.row[filerow].rsize - E.coloff;

if (len < 0)

len = 0;

if (len > E.screencols)

len = E.screencols;

char \*c = &E.row[filerow].render[E.coloff];

unsigned char \*hl = &E.row[filerow].hl[E.coloff];

int current\_color = -1;

int j;

for (j = 0; j < len; j++) {

if (iscntrl(c[j])) {

char sym = (c[j] <= 26) ? '@' + c[j] : '?';

abAppend(ab, "\x1b[7m", 4);

abAppend(ab, &sym, 1);

abAppend(ab, "\x1b[m", 3);

if (current\_color != -1) {

char buf[16];

int clen = snprintf(buf, sizeof(buf), "\x1b[%dm", current\_color);

abAppend(ab, buf, clen);

}

} else if (hl[j] == HL\_NORMAL) {

if (current\_color != -1) {

abAppend(ab, "\x1b[39m", 5);

current\_color = -1;

}

abAppend(ab, &c[j], 1);

} else {

int color = editorSyntaxToColor(hl[j]);

if (color != current\_color) {

current\_color = color;

char buf[16];

int clen = snprintf(buf, sizeof(buf), "\x1b[%dm", color);

abAppend(ab, buf, clen);

}

abAppend(ab, &c[j], 1);

}

}

abAppend(ab, "\x1b[39m", 5);

}

abAppend(ab, "\x1b[K", 3);

abAppend(ab, "\r\n", 2);

}

}

void editorDrawStatusBar(struct abuf \*ab) {

abAppend(ab, "\x1b[7m", 4);

char status[80], rstatus[80];

int len = snprintf(status, sizeof(status), "%.20s - %d lines %s",

E.filename ? E.filename : "[No Name", E.numrows,

E.dirty ? "(modified)" : "");

int rlen =

snprintf(rstatus, sizeof(rstatus), "%s | %d/%d",

E.syntax ? E.syntax->filetype : "no ft", E.cy + 1, E.numrows);

if (len > E.screencols)

len = E.screencols;

abAppend(ab, status, len);

while (len < E.screencols) {

if (E.screencols - len == rlen) {

abAppend(ab, rstatus, rlen);

break;

} else {

abAppend(ab, " ", 1);

len++;

}

}

abAppend(ab, "\x1b[m", 3);

abAppend(ab, "\r\n", 2);

}

void editorDrawMessageBar(struct abuf \*ab) {

abAppend(ab, "\x1b[K", 3);

int msglen = strlen(E.statusmsg);

if (msglen > E.screencols)

msglen = E.screencols;

if (msglen && time(NULL) - E.statusmsg\_time < 5)

abAppend(ab, E.statusmsg, msglen);

}

void editorRefreshScreen() {

editorScroll();

struct abuf ab = ABUF\_INIT;

abAppend(&ab, "\x1b[?25l", 6);

abAppend(&ab, "\x1b[H", 3);

editorDrawRows(&ab);

editorDrawStatusBar(&ab);

editorDrawMessageBar(&ab);

char buf[32];

snprintf(buf, sizeof(buf), "\x1b[%d;%dH", (E.cy - E.rowoff) + 1,

(E.rx - E.coloff) + 1);

abAppend(&ab, buf, strlen(buf));

abAppend(&ab, "\x1b[?25h", 6);

write(STDOUT\_FILENO, ab.b, ab.len);

abFree(&ab);

}

void editorSetStatusMessage(const char \*fmt, ...) {

va\_list ap;

va\_start(ap, fmt);

vsnprintf(E.statusmsg, sizeof(E.statusmsg), fmt, ap);

va\_end(ap);

E.statusmsg\_time = time(NULL);

}

char \*editorPrompt(char \*prompt, void (\*callback)(char \*, int)) {

size\_t bufsize = 128;

char \*buf = malloc(bufsize);

size\_t buflen = 0;

buf[0] = '\0';

while (1) {

editorSetStatusMessage(prompt, buf);

editorRefreshScreen();

int c = editorReadKey();

if (c == DEL\_KEY || c == CTRL\_KEY('h') || c == BACKSPACE) {

if (buflen != 0)

buf[--buflen] = '\0';

} else if (c == '\x1b') {

editorSetStatusMessage("");

if (callback)

callback(buf, c);

free(buf);

return NULL;

} else if (c == '\r') {

if (buflen != 0) {

editorSetStatusMessage("");

if (callback)

callback(buf, c);

return buf;

}

} else if (!iscntrl(c) && c < 128) {

if (buflen == bufsize - 1) {

bufsize \*= 2;

buf = realloc(buf, bufsize);

}

buf[buflen++] = c;

buf[buflen] = '\0';

}

if (callback)

callback(buf, c);

}

}

void editorMoveCursor(int key) {

erow \*row = (E.cy >= E.numrows) ? NULL : &E.row[E.cy];

switch (key) {

case ARROW\_LEFT:

if (E.cx != 0) {

E.cx--;

} else if (E.cy > 0) {

E.cy--;

E.cx = E.row[E.cy].size;

}

break;

case ARROW\_RIGHT:

if (row && E.cx < row->size) {

E.cx++;

} else if (row && E.cx == row->size) {

E.cy++;

E.cx = 0;

}

break;

case ARROW\_UP:

if (E.cy != 0) {

E.cy--;

}

break;

case ARROW\_DOWN:

if (E.cy != E.numrows) {

E.cy++;

}

break;

}

row = (E.cy >= E.numrows) ? NULL : &E.row[E.cy];

int rowlen = row ? row->size : 0;

if (E.cx > rowlen) {

E.cx = rowlen;

}

}

void editorProcessKeypress() {

static int quit\_times = KILO\_QUIT\_TIMES;

int c = editorReadKey();

switch (c) {

case '\r':

editorInsertNewLine();

break;

case CTRL\_KEY('q'):

if (E.dirty && quit\_times > 0) {

editorSetStatusMessage("WARNING!!! File has unsaved changes. "

"Press Ctrl-Q %d more times to quit.",

quit\_times);

quit\_times--;

return;

}

write(STDOUT\_FILENO, "\x1b[2J", 4);

write(STDOUT\_FILENO, "\x1b[H", 3);

exit(0);

break;

case CTRL\_KEY('s'):

editorSave();

break;

case HOME\_KEY:

E.cx = 0;

break;

case END\_KEY:

if (E.cy < E.numrows)

E.cx = E.row[E.cy].size;

break;

case CTRL\_KEY('f'):

editorFind();

break;

case BACKSPACE:

case CTRL\_KEY('h'):

case DEL\_KEY:

if (c == DEL\_KEY)

editorMoveCursor(ARROW\_RIGHT);

editorDelChar();

break;

case PAGE\_UP:

case PAGE\_DOWN: {

if (c == PAGE\_UP) {

E.cy = E.rowoff;

} else if (c == PAGE\_DOWN) {

E.cy = E.rowoff + E.screenrows - 1;

if (E.cy > E.numrows)

E.cy = E.numrows;

}

int times = E.screenrows;

while (times--)

editorMoveCursor(c == PAGE\_UP ? ARROW\_UP : ARROW\_DOWN);

} break;

case ARROW\_UP:

case ARROW\_DOWN:

case ARROW\_LEFT:

case ARROW\_RIGHT:

editorMoveCursor(c);

break;

case CTRL\_KEY('l'):

case '\x1b':

break;

default:

editorInsertChar(c);

break;

}

quit\_times = KILO\_QUIT\_TIMES;

}

void initEditor() {

E.cx = 0;

E.cy = 0;

E.rx = 0;

E.rowoff = 0;

E.coloff = 0;

E.numrows = 0;

E.row = NULL;

E.dirty = 0;

E.filename = NULL;

E.statusmsg[0] = '\0';

E.statusmsg\_time = 0;

E.syntax = NULL;

if (getWindowSize(&E.screenrows, &E.screencols) == -1)

die("getWindowSize");

E.screenrows -= 2;

}

int main(int argc, char \*argv[]) {

enableRawMode();

initEditor();

if (argc >= 2) {

editorOpen(argv[1]);

}

editorSetStatusMessage("HELP: CTRL-S = save | Ctrl-Q = quit | Ctrl-F = find");

while (1) {

editorRefreshScreen();

editorProcessKeypress();

}

return 0;

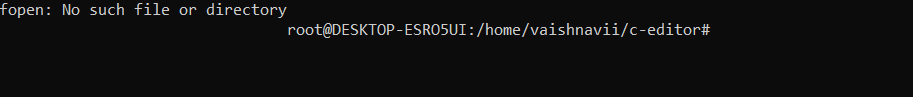
}

**Github repository**

<https://github.com/kothavaishnavi2002/Miniproject>

This repository contains abstract, design document, source code and report for text editor project.

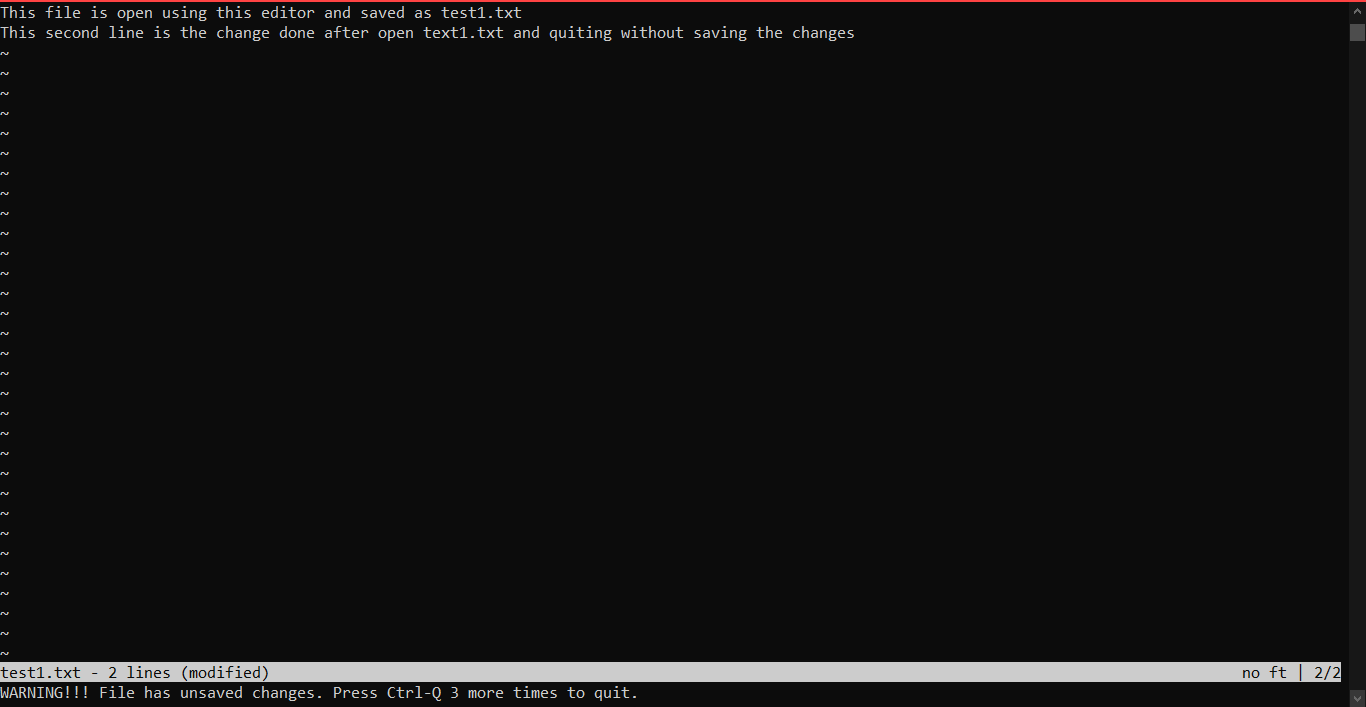
**C. Testing**



When we try to open a file that doesn't exist using a command prompt, the output says no file.

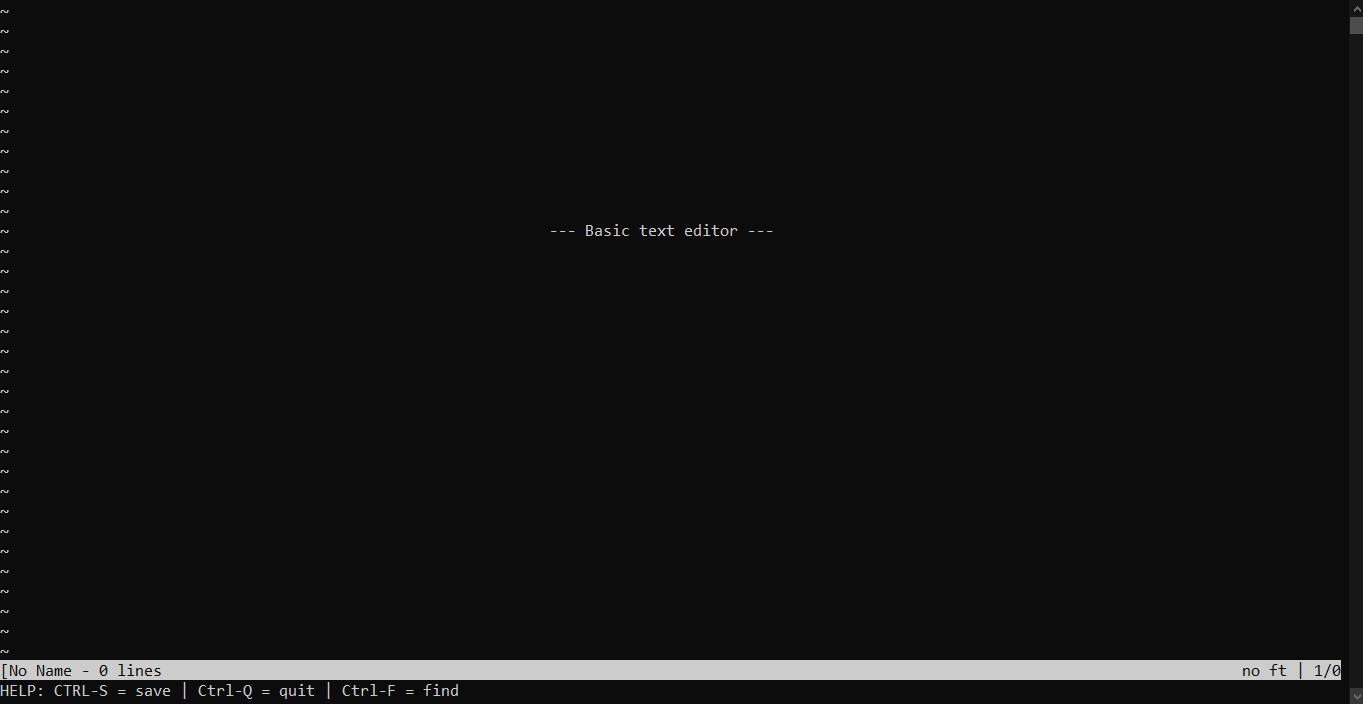


When we execute this program and quit without saving the changes the warnings are shown and we need to press ctrl-Q 3 more times. Since this is not an existing file, it says [No Name - 1 lines (modified)]. Modified indicated it has unsaved changes.

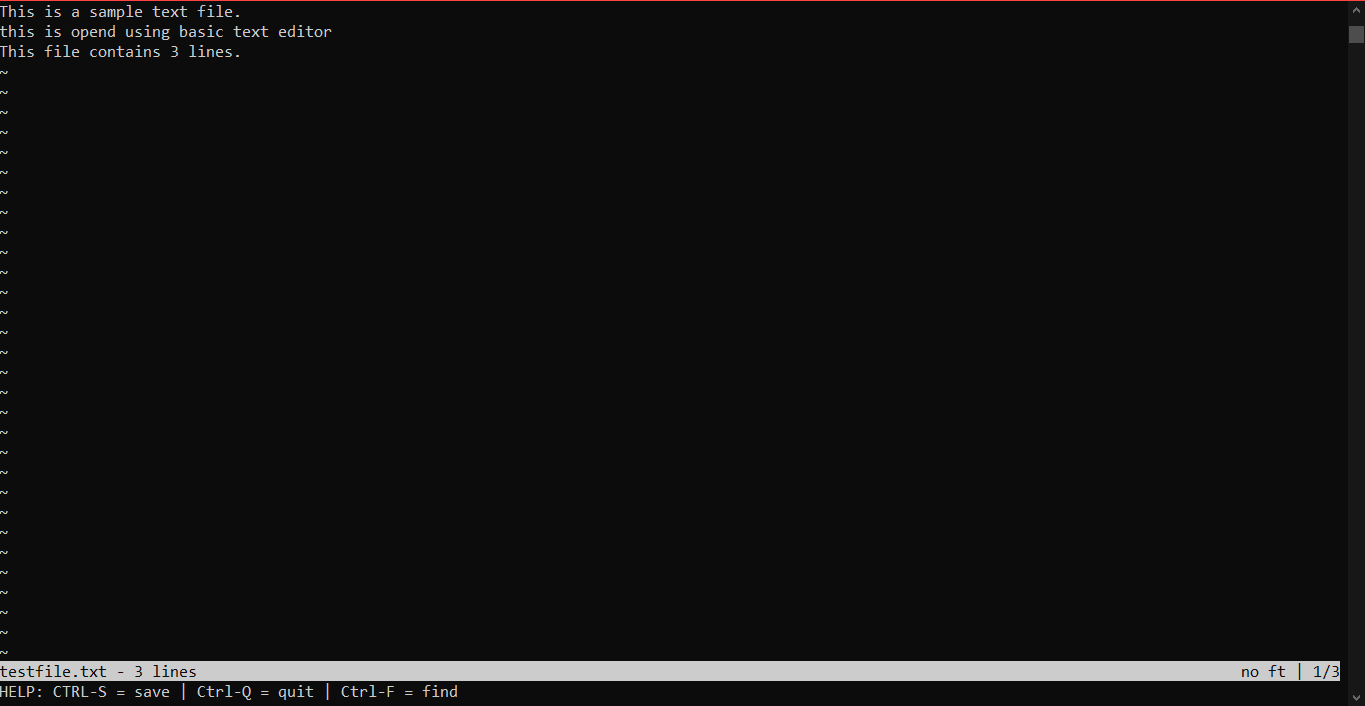


When the existing file is opened and when tried to quit, a warning is shown saying filename - (modified). Which means it has unsaved changes.

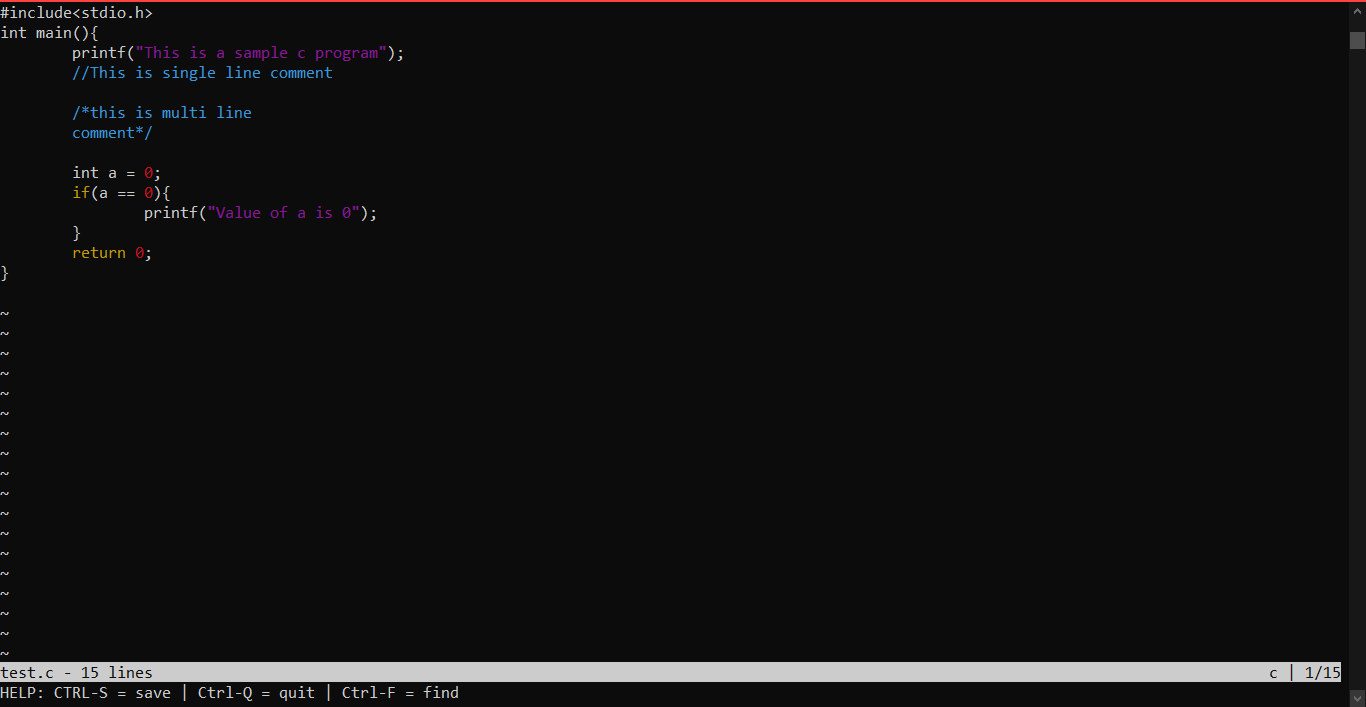
**Results**



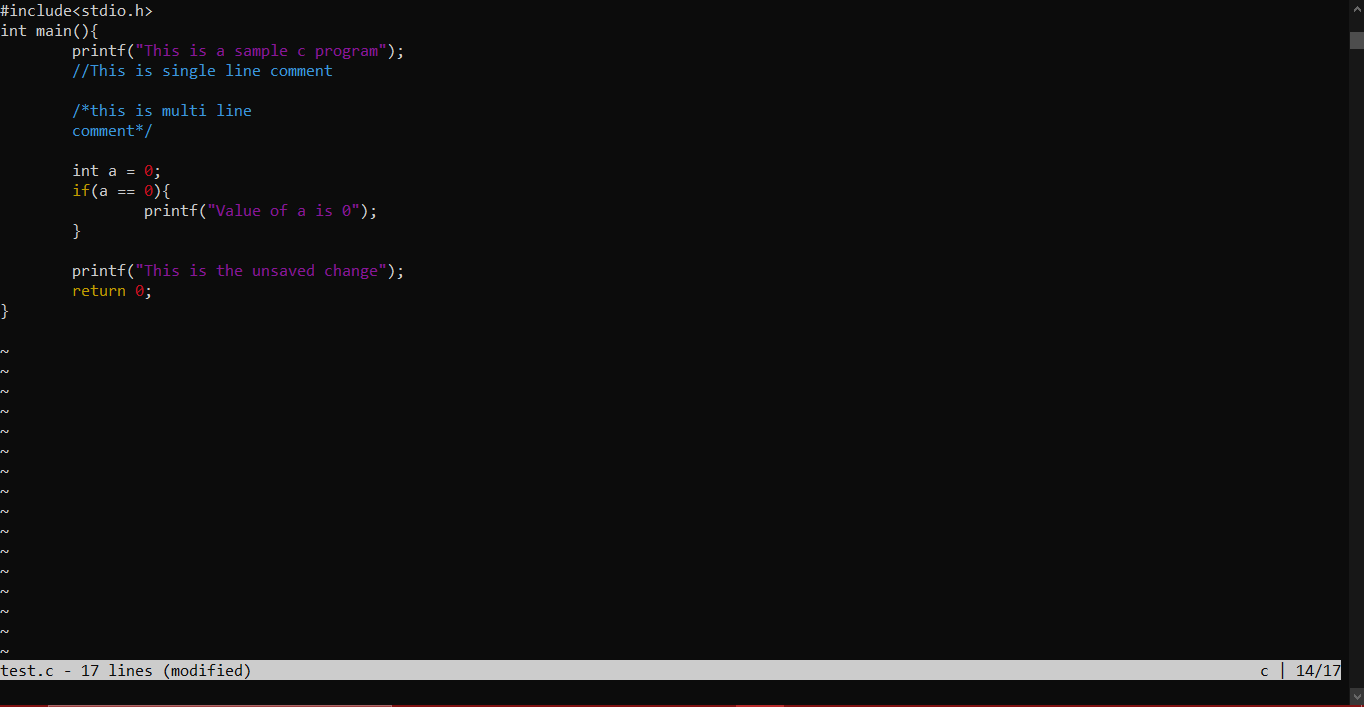
This is the output of the execution of the text editor. There is a prompt showing “---Basi text editor ---” when we execute without opening any file. It also shows “No Name“ because it isn’t saved. And below are some of the keys for knowing how to save, exit, and quit.



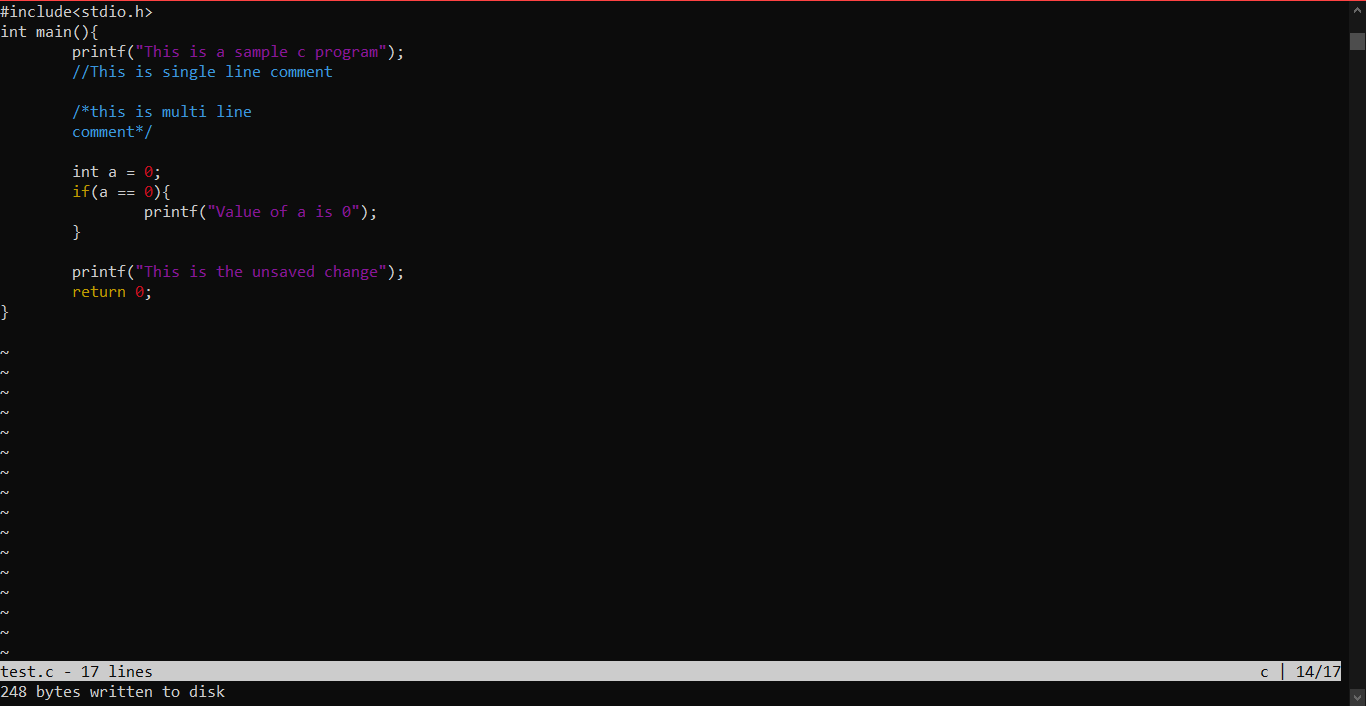
This is the output when we open a text file. The file name is shown at the bottom. And now we cannot see the prompt saying “ ---basic text editor --- ”. It also shows the number of lines in the given file.



This gets displayed when we try to open a .c file. Keywords and comments are written in a different color which shows the syntax highlighting.



When changes are done after opening the page, it shows “(modified)”



After saving the changes done it says how many bytes are saved. And now we can quit by just pressing ctrl - Q once.

**Additional learning**

The way we learned to manage our time. Basically, time management of classes, assignments, quizzes, and other stuff was great learning not only professionally but also helped both of us to grow as individuals. This project helped both of us to put our interest in coding in a certain work and really helped it to grow. Not only time but also the workload had to be managed a lot because some of the assignments weren’t that time taking but were mentally really exhausting. Both of us had an amazing experience working together and we really understood the importance of teamwork which is something that helps you to increase your spread of thoughts due to various opinions help the work to complete in less time, in a more efficient manner.

Self-learning is another important aspect which we needed to explore in ourselves as there were few things which we didn’t know and we needed to take the help of google and needed to learn them by ourselves online. And this made the understanding of our project a lot easier and helped us to be more creative in various steps of its development. We also had to revise a lot of concepts regarding programming in C, which made our basics even stronger and is helping us to be even more confident. We all knew that we use a text editor to generally write a program before compiling it. We were always keen on how a program is written and how it functions. This was the most interesting part of the project. We saw this as an opportunity to improve our skills rather than an assigned task which helped us to complete this project with more excitement and enthusiasm.

**Discussion and Future Work**

We have a lot of plans that we would like to add a lot of elements to our project. We would like to add more file types like add syntax highlighting rules for some of our favorite languages to the HLDB array, things like line number i.e display the line number to the left of each line of the file and the soft indent which is helpful if you like using spaces instead of tabs, make the Tab key insert spaces instead of ‘\t’, Interesting things like auto-indent which will be in a starting of the new line as well as the previous line. The hard-wrap lines, soft-wrap lines, and usage of cursors are going to be some of the coolest features.

The copy and paste addition would be a really very helpful addition, it would give the user a way to select text and then copy the selected text when they press ctrl+c and let them paste when they press ctrl+v.

Modal editing is an interesting addition i.e if you like vim, make this text editor work more like vim by letting the user press ‘i’ for insert mode and then press ‘ESC’ to escape and go back to normal mode. Then start adding all your favorite vim commands, starting with basic movement commands.

We would also love to try to use the functioning of multiple files open at once, and have some way of switching between them.

**References**

We have a booklet by referring that we learned how to build a text editor.

<https://viewsourcecode.org/snaptoken/kilo/>