**TYPEAWAY: A Linux Text Editor**

*A Mini Project Report*

*Submitted in partial fulfilment of the*

*Requirements for the award of the Degree of*

**BACHELOR OF ENGINEERING**

IN

**INFORMATION TECHNOLOGY**

By

**N. TARUNI 1602-19-737-120**

**J . PRANAVI 1602-19-737-184**

****

**Department of Information Technology**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Ibrahimbagh, Hyderabad-31**

**2020**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Hyderabad-500 031**

**Department of Information Technology**

****

**DECLARATION BY THE CANDIDATE**

We, **N TARUNI** **and J . PRANAVI** bearing hall ticket numbers, 1602-19-737-120, and 1602-19-737-184 respectively, hereby declare that the project report entitled **TYPEAWAY: A Linux Text Editor** is submitted in partial fulfilment 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 us and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

**N TARUNI**

**1602-19-737-120**

**J PRANAVI**

**1602-19-737-184**

(Faculty In-Charge) (Head, Dept. of IT)

**Acknowledgment**

Our Mini Project would not have been successful without the help of several people. We are extremely thankful to our college, **Vasavi College of Engineering, Hyderabad** for providing the opportunity to implement our project, **“TYPEAWAY: A Linux Text Editor”**.

We would like to express our gratitude to **Ms. DRL Prasanna**, Assistant Professor, Department of Information Technology, Vasavi College of Engineering and **Dr. K Ram Mohan Rao**, Professor and HOD, Department of Information Technology, Vasavi College of Engineering, for their esteemed guidance, moral support and invaluable advice provided by them for the success of the Mini Project.

Sincerely,

**N TARUNI 1602-19-737-120**

**J PRANAVI 1602-19-737-184**

## 

## ABSTRACT

Type Away, as the name suggests, is the recreation of a linux text editor, made using C language. It has exciting features like search, syntax highlighting and adding notes to your text file. The UI is very simple and the colours used catch the eye immediately. This editor allows the user to open, read and edit new and existing documents. The user can also search through the documents. We have also included an additional feature to write notes in a text file.

**GitHub Links:**

<https://github.com/Pranavi112>

<https://github.com/taruni-always>

**Team Number:** 28

**Team members:**

N Taruni (1602-19-737-120)

Jamalapuram Pranavi (1602-19-737-184)

**TABLE OF CONTENTS**

# 

[**1. INTRODUCTION**](#_ja10uikbcvlj) **07**

[1.1. ABOUT THE PROJECT](#_uc26dil214v8) 07

[1.2. PROJECT DOMAIN](#_c83m3n6g01zj) 07

[1.2.1. TECHNICAL DOMAIN](#_8ms9zqdtxgyv) 07

[1.2.2. FUNCTIONAL DOMAIN](#_escfs61npun7) 07

[1.3. FEATURES](#_ihku94sumas5) 07

[**2. TECHNOLOGY**](#_mmz4gq5jt8w2) **08**

[**3. PROPOSED WORK**](#_k2n4o69mr9y1) **09**

[**3.1. DESIGN**](#_vrsv1nymjfbs) **09**

[USER USE CASES](#_mvuwkrq72xs4) 09

[3.1.1.1. OPEN DOCUMENT](#_id0sua4romyd) 09

[3.1.1.2. QUIT DOCUMENT](#_felo4q2dy7ui) 09

[3.1.1.3. SAVE DOCUMENT](#_quy9enucoa55) 10

[3.1.1.4. SYNTAX HIGHLIGHTING](#_2z9wfd6z68yp) 10

[3.1.1.5. SEARCH](#_yyve6ubkwuzu) 10

[**3.2. IMPLEMENTATION**](#_hcu2stmi86l6) **10**

[3.2.1. CODE](#_w14an6rpm4nn) 10

[3.2.2. GITHUB/FOLDER STRUCTURE](#_sa3on5tdwupt) 41

[**3.3. TESTING:**](#_v5jvyhy3sp77) **42**

[USER TEST CASES:](#_wy8rkh7bmxqz) 42

[3.3.1. OPEN DOCUMENT](#_s9hbegve3lg3) 43

[3.3.3. SAVE DOCUMENT](#_l4odtr56kylk) 45

[3.3.4. SYNTAX HIGHLIGHTING](#_hbmni1j48zku) 46

[3.3.5. SEARCH](#_lrbpw7j0azv2) 46

[**4. RESULTS**](#_82d28f2raii2) **48**

[USER TEST CASE RESULTS](#_iiviuhawl7ok) 48

[Test Case 1: Open Existing Document](#_gsg42byt1x0d) 48

[Test Case 2: Open New Document](#_nmkxhyz1prst) 48

[Test Case 3: Quitting Before Saving](#_wdxupomwzh82) 49

[Test Case 4: Quitting After Saving](#_icp9ckr4sp26) 50

[Test Case 5: Saving Changes Made To An Existing Document](#_l63w282kzthh) 51

[Test Case 6: Saving Changes In A New Document](#_l4rhse8qib44) 51

[Test Case 7: Syntax Highlighting](#_5gocq4p0c7vv) 52

[Test Case 8: Searching For A word/words](#_xlqa4396icss) 53

[**5. ADDITIONAL KNOWLEDGE ACQUIRED**](#_uhhoec3zfruc) **54**

[**6. CONCLUSION AND FUTURE WORK**](#_kg8x9flxwjeu) **55**

[**7. REFERENCES**](#_daq9jj3yrnj5) **56**

# 

# 1. INTRODUCTION

## 1.1. ABOUT THE PROJECT

## TypeAway is a linux text editor. It’s about 1000 lines of C in a single file with no dependencies, and it implements all the basic features you expect in a minimal editor, as well as syntax highlighting and a search feature.

## 1.2. PROJECT DOMAIN

The domain of the project is the targeted subject area of a computer program . It is a term most commonly used in software engineering. Formally, it represents the target subject of a specific programming project, whether narrowly or broadly defined. To be concise, a domain in the realm of software engineering commonly refers to the subject area on which the application is intended to apply. Domain consist of two categories:

1. Technical Domain

2. Functional Dom

### 1.2.1. TECHNICAL DOMAIN

" TypeAway" is a console based C project. It comes under the domain of a console application. A console application is a program designed to be used via a text-only computer interface, such as a text terminal, the command line interface of some operating systems or the text-based interface included with most GUI (Graphical User Interface) operating systems.

### 1.2.2. FUNCTIONAL DOMAIN

### This project is a simple linux text editor which can perform all the functions of a text editor. In addition to this it has syntax highlighting and search features.

## 1.3. FEATURES

The main features of our text editor are:

→ **Colour:** We have used vibrant colours to convey meaning beyond the basic text.

→ **Syntax highlighting:** It helps to make the code more readable, especially when it's in the context of a document full of other kinds of text.

→ **Notes:** The user can take notes in a text document/file. In order to make a note, the user needs to type “note:” and the following sentence becomes highlighted as a note throughout the document.

# 2. TECHNOLOGY

In every computer software we need certain hardware components or other software resources to be present on a computer. These prerequisites are known as system requirements. We have two types - Software Requirements and hardware Requirements.

The entire project was made in C language. We used the built-in libraries in C to run our project smoothly.

**2.1 Software requirements:**

Software Requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

The required software Requirements for our project:

→ **Operating system:** Linux Operating System or WSL (Windows subsystem for linux),

→ **C Compiler**: GNU Compiler Collection (GCC)

**2.1 Hardware requirements:**

Hardware Requirements defined by any operating system or software application is the physical computer resources.

The required Hardware Requirements for our project:

→ **Processor**: Intel Core i5 and above

→ **Memory**: 8 GB

# 

# 3. PROPOSED WORK

## 3.1. DESIGN

The user has access to the 5 functionalities: Open Document, Quit Document, Save Document, Syntax Highlighting and Search.

### USER USE CASES

## 

#### 3.1.1.1. OPEN DOCUMENT

The user can open either new documents or existing documents. If the user specifies the filename, that existing document is opened for the user to edit and write in. If the user does not specify any filename, a new document is loaded for the user to write in.

#### 3.1.1.2. QUIT DOCUMENT

The user can exit/quit the opened document. If the user hasn’t made any changes, the document can directly be closed. If the user has made some modifications to the document, the user needs to save the document for the changes to be reflected, before quitting the document. Otherwise, if the user does not wish the changes to be reflected, then he/she can quit the document.

#### 3.1.1.3. SAVE DOCUMENT

The save option is for the changes to be reflected when the file is opened next time. In case the opened document is an existing one, he/she can directly save the document. If the opened document is a new one, the user is prompted to enter a filename to save the current document as.

#### 3.1.1.4. SYNTAX HIGHLIGHTING

The feature of syntax highlighting is only applicable when the file has been saved at least once. When a file is saved, the system detects it’s filetype and automatically highlights the necessary syntax. Therefore it is important for the user to save the document at least once. In case of a C file, the special keywords like “int”, “main” are highlighted along with single line and multiline comments. In the case of text files, only numbers are highlighted.

#### 3.1.1.5. SEARCH

We have implemented an incremental search feature to progressively **search** for and filter through text. As the user types text, one or more possible matches for the text are found and immediately presented to the user. The user can use arrow keys to navigate through multiple occurrences of the word. The user can press the “ENTER” key to stop the search and point to the word, or press the “ESC” key to exit from search.

## 3.2. IMPLEMENTATION

### 3.2.1. CODE

#define \_DEFAULT\_SOURCE

#define \_BSD\_SOURCE

#define \_GNU\_SOURCE

#define TAB\_STOP 4

/\*\*\* Include statements\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>//the POSIC Operating System API

#include <termios.h> //for terminal I/O interface

#include <ctype.h> //for iscntrl() method

#include <errno.h> //for handling errors

#include <sys/ioctl.h> //to get terminal dimensions

#include <string.h>

#include <time.h>

#include <stdarg.h>

#include <fcntl.h>

//#include "search.h"

/\*\*\* defining our own macros\*\*\*/

#define CTRL\_KEY(key) ((key) & 0x1f) // ANDing with 31 i.e 1f in hexadecimal ex: 'a' - 97, 'a' & 0x1f - 1

#define ABUF\_INIT {NULL, 0}

enum keys {

BACK\_SPACE = 127,

ARROW\_LEFT = 1000,

ARROW\_RIGHT, ARROW\_UP,

ARROW\_DOWN,

DEL\_KEY,

HOME\_KEY,

END\_KEY,

PAGE\_UP,

PAGE\_DOWN

};

enum highlight {

HL\_NORMAL = 0,

HL\_COMMENT,

HL\_MLCOMMENT,

HL\_KEYWORD1,

HL\_KEYWORD2,

HL\_STRING,

HL\_NUMBER,

HL\_TEXT,

HL\_MATCH

};

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

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

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

/\*\*\*Data\*\*\*/

struct editorSyntax {

char \*fileType;

char \*\*fileMatch;

char \*\*keywords;

char \*singleLineCommentStart;

char \*multiLineCommentsStart;

char \*multiLineCommentsEnd;

int flags;

};

typedef struct editorRow {

int index;

int size, rsize;

char \*chars;

char \*render;

char \*hl; //highlighting

int hlOpenComment;

} editorRow;

/\*\*\* global variables \*\*\*/

struct configurations {

int xCoord, yCoord;

int rx;

int rowOffset, colOffset;

int terminalRows, terminalCols;

int numrows;

editorRow \*row;

int dirty;// to know if the changes are saved or not

char \*fileName;

char statusmsg[80];

time\_t statusmsg\_time;

struct editorSyntax \*syntax;

struct termios originalTerminal;

};

struct configurations editor;

/\*\*\*file types\*\*\*/

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

char \*TEXT\_HL\_extension[] = {".txt", ".docx", NULL};

char \*TEXT\_HL\_keywords[] = {"", NULL};

struct editorSyntax HLDB[] = { // highlight database

{

"c/c++",

C\_HL\_extensions,

C\_HL\_keywords,

"//", "/\*", "\*/",

HL\_HIGHLIGHT\_NUMBERS | HL\_HIGHLIGHT\_STRINGS

},

{

"text",

TEXT\_HL\_extension,

TEXT\_HL\_keywords,

"note:", "", "",

HL\_HIGHLIGHT\_NUMBERS

}

};

#define HLDB\_ENTRIES (sizeof(HLDB) / sizeof(HLDB[0]))

/\*\*\* append buffer \*\*\*/

struct abuf {

char \*b;

int len;

};

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);

}

/\*\*\*prototypes\*\*\*/

int xCoordTorx(editorRow \*row, int cx) {

int rx = 0;

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

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

rx += (TAB\_STOP - 1) - (rx % TAB\_STOP);

rx ++;

}

return rx;

}

int rxToxCoord(editorRow \*row, int rx) {

int currentRX = 0, x;

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

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

currentRX += (TAB\_STOP - 1) - (currentRX % TAB\_STOP);

currentRX ++;

if (currentRX > rx) return x;

}

return x;

}

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

char \*prompt(char \*message, void (\*callback)(char \*, int));

int colourCodes(int hl);

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

void editorScroll() {

editor.rx = 0;

if (editor.yCoord < editor.numrows) {

editor.rx = xCoordTorx(&editor.row[editor.yCoord], editor.xCoord);

}

if (editor.yCoord < editor.rowOffset) {

editor.rowOffset = editor.yCoord;

}

if (editor.yCoord >= editor.rowOffset + editor.terminalRows) {

editor.rowOffset = editor.yCoord - editor.terminalRows + 1;

}

if (editor.rx < editor.colOffset) {

editor.colOffset = editor.rx;

}

if (editor.rx >= editor.colOffset + editor.terminalCols) {

editor.colOffset = editor.rx - editor.terminalCols + 1;

}

}

void indicateRows(struct abuf \*ab) {

for (int currRow = 0; currRow < editor.terminalRows; currRow ++) {

int fileRow = currRow + editor.rowOffset;

if (fileRow >= editor.numrows) {

if (editor.numrows == 0 && currRow == editor.terminalRows / 3) {

char welcome[80];

int welcomelen = snprintf(welcome, sizeof(welcome), "\x1b[33m T\x1b[35my\x1b[33mP\x1b[35me Away!!\x1b[m");

if (welcomelen > editor.terminalCols) welcomelen = editor.terminalCols;

int padding = (editor.terminalCols - welcomelen) / 2;

if (padding) {

abAppend(ab, "~", 1);//cyan

padding--;

}

while (padding--) abAppend(ab, " ", 1);

abAppend(ab, welcome, welcomelen);

}

else {

abAppend(ab, "~", 1);//light blue

}

}

else {

int len = editor.row[fileRow].rsize - editor.colOffset;

if (len < 0) len = 0;

if (len > editor.terminalCols) len = editor.terminalCols;

char \*c = &editor.row[fileRow].render[editor.colOffset];

char \*hl = &editor.row[fileRow].hl[editor.colOffset];

int currentColour = -1;

for (int 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 (currentColour != -1) {

char buf[16];

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

abAppend(ab, buf, clen);

}

}

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

if (currentColour != -1) {

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

currentColour = -1;

}

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

}

else {

int colour = colourCodes(hl[j]);

if (colour != currentColour) {

currentColour = colour;

char buf[16];

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

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 drawStatusBar(struct abuf \*ab) {

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

char status[80], rstatus[80];

int len = snprintf(status, sizeof(status), "\x1b[35m %.20s - %d lines %s\x1b[m", editor.fileName ?

editor.fileName : "[Unknown File]", editor.numrows, editor.dirty ? "(modified)" : "");

int rlen = snprintf(rstatus, sizeof(rstatus), "%s | %d/%d", editor.syntax ?

editor.syntax -> fileType : "no file type", editor.yCoord + 1, editor.numrows);

if (len > editor.terminalCols) len = editor.terminalCols;

if (len > editor.terminalCols) len = editor.terminalCols;

abAppend(ab, status, len);

while (len < editor.terminalCols) {

if (editor.terminalCols - len == rlen) {

abAppend(ab, rstatus, rlen);

break;

}

else {

abAppend(ab, " ", 1);

len ++;

}

}

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

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

}

void setStatusMessage( const char \*fmt, ...) {//variable number of arguements

va\_list ap;

//strcat(fmt, "\x1b[32m");

va\_start(ap, fmt);

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

va\_end(ap);

editor.statusmsg\_time = time(NULL);

}

void drawMessageBar(struct abuf \*ab) {

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

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

int msgLen = strlen(editor.statusmsg);

if ( msgLen > editor.terminalCols) msgLen = editor.terminalCols;

if ( msgLen && time(NULL) - editor.statusmsg\_time < 5) abAppend(ab, editor.statusmsg, msgLen);

}

void refreshScreen() {

editorScroll();

struct abuf ab = ABUF\_INIT;

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

//abAppend(&ab, "\x1b[2J", 4);

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

indicateRows(&ab);

drawStatusBar(&ab);

drawMessageBar(&ab);

char buf[32];

snprintf(buf, sizeof(buf), "\x1b[%d;%dH", editor.yCoord - editor.rowOffset + 1, editor.rx - editor.colOffset + 1);

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

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

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

abFree(&ab);

}

/\*\*\* Terminal \*\*\*/

void handleError(const char \*s) {

refreshScreen();

perror(s);

exit(1);

}

void turnOffFlags(struct termios raw) {

raw.c\_iflag &= ~(IXON | ICRNL | INPCK | BRKINT); //'~' complementing and then '&' ANDing the bits

raw.c\_oflag &= ~(OPOST); //'~' complementing and then '&' ANDing the bits

raw.c\_cflag |= ~(CS8); //ORing this time and not ANDing

raw.c\_lflag &= ~(ECHO | ICANON | ISIG | IEXTEN); //'~' complementing and then '&' ANDing the bits

//turning off a few needed flags

}

void disableRawMode() {

tcsetattr(STDIN\_FILENO, TCSAFLUSH, &editor.originalTerminal);

}

void enableRawMode() {

struct termios raw = editor.originalTerminal;

atexit(disableRawMode);

if ( tcgetattr(STDIN\_FILENO, &(editor.originalTerminal)) == -1) handleError("tcgetattr");

turnOffFlags(raw);

raw.c\_cc[VMIN] = 0;

raw.c\_cc[VTIME] = 1;

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

} // function to enable raw mode

int readKey() {

int nread;

char c;

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

if (nread == -1 && errno != EAGAIN) handleError("read");

}

if (c == '\x1b') { //arrow keys have the escape sequence '\x1b' at the beginning

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] == 'O') {

switch (seq[1]) {

case 'H': return HOME\_KEY;

case 'F': return END\_KEY;

}

}

return '\x1b';

}

else {

return c;

}

} //separate function because we 're processing it only after we read a valid key w/o errors

int getCursorPosition(int \*rSize, int \*cSize) {

char buffer[32];

unsigned int i = 0;

if (write(STDOUT\_FILENO, "\x1b[6n", 4) != 4) return -1;

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

if (read(STDIN\_FILENO, &buffer[i], 1) != 1) break;

if (buffer[i] == 'R') break;

i ++;

}

buffer[i] = '\0';

if (buffer[0] != '\x1b' || buffer[1] != '[') return -1;

if (sscanf( &buffer[2], "%d;%d", rSize, cSize) != 2) return -1;

return 0;

}

int getWindowSize(int \*rSize, int \*cSize) {

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(rSize, cSize);

}

else {

\*cSize = ws.ws\_col;

\*rSize = ws.ws\_row;

return 0;

}

}

/\*\*\* syntax highlighting \*\*\*/

int isSeparator(int c) {

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

}

int colourCodes(int hl) {

switch (hl) {

case HL\_MLCOMMENT:

case HL\_COMMENT: return 36; //cyan

case HL\_KEYWORD1: return 33; //Brown

case HL\_KEYWORD2: return 32; // Green

case HL\_NUMBER: return 31; //red

case HL\_TEXT: return 33;

case HL\_MATCH: return 32; //green // when we found the search results

case HL\_STRING: return 34; //Blue

default: return 37;

}

}

void updateSyntax(editorRow \*row) {

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

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

if (editor.syntax == NULL) return;

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

char \*scStart = editor.syntax -> singleLineCommentStart;

char \*mcStart = editor.syntax -> multiLineCommentsStart;

char \*mcEnd = editor.syntax -> multiLineCommentsEnd;

int scStartLen = scStart ? strlen(scStart) : 0;

int mcStartLen = mcStart ? strlen(mcStart) : 0;

int mcEndLen = mcEnd ? strlen(mcEnd) : 0;

int prevSeperator = 1;

int inString = 0;

int inComment = (row -> index > 0 && editor.row[row -> index - 1].hlOpenComment);

int i = 0;

while (i < row->rsize) {

char c = row -> render[i];

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

if (scStartLen && !inString && !inComment) {

if (!strncmp(&row->render[i], scStart, scStartLen)) {

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

break;

}

}

if (mcStartLen && mcEndLen && !inString) {

if (inComment) {

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

if (!strncmp(&row -> render[i], mcEnd, mcEndLen)) {

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

i += mcEndLen;

inComment = 0;

prevSeperator = 1;

continue;

}

else {

i ++;

continue;

}

}

else if (!strncmp(&row -> render[i], mcStart, mcStartLen)) {

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

i += mcStartLen;

inComment = 1;

continue;

}

}

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

if (inString) {

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

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

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

i += 2;

continue;

}

if (c == inString) inString = 0;

i ++;

prevSeperator = 1;

continue;

}

else {

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

inString = c;

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

i ++;

continue;

}

}

}

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

if ( (isdigit(c) && (prevSeperator || prevhl == HL\_NUMBER)) || (c == '.' && prevhl == HL\_NUMBER)) { //decimal numbers also

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

i ++;

prevSeperator = 0;

continue;

}

}

if (editor.syntax -> flags & HL\_HIGHLIGHT\_TEXT) {

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

}

if (prevSeperator) {

int j;

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

int klen = strlen(keywords[j]);

int keword2 = keywords[j][klen - 1] == '|';

if (keword2) klen --;

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

memset(&row->hl[i], keword2 ? HL\_KEYWORD2 : HL\_KEYWORD1, klen);

i += klen;

break;

}

}

if (keywords[j] != NULL) {

prevSeperator = 0;

continue;

}

}

prevSeperator = isSeparator(c);

i ++;

}

int changed = (row -> hlOpenComment != inComment);

row -> hlOpenComment = inComment;

if (changed && row -> index + 1 < editor.numrows)

updateSyntax(&editor.row[row -> index + 1]);

}

void selectSyntaxHighlight() {

editor.syntax = NULL;

if (editor.fileName == NULL) return;

char \*ext = strrchr(editor.fileName, '.');

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

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

int i = 0;

while (s -> fileMatch[i]) {

int isExtension = (s -> fileMatch[i][0] == '.');

if ((isExtension && ext && !strcmp(ext, s -> fileMatch[i])) || (!isExtension && strstr(editor.fileName, s -> fileMatch[i]))) {

editor.syntax = s;

for ( int fileRow = 0; fileRow < editor.numrows; fileRow ++) {

updateSyntax(&editor.row[fileRow]);

}

return;

}

i++;

}

}

}

/\*\*\*manipulating row actions\*\*\*/

void updateRow(editorRow \*row) {

free(row->render);

row->render = malloc(row->size + 1);

int index = 0, tabs = 0;

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

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

}

free(row -> render);

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

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

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

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

while (index % TAB\_STOP != 0) row->render[index ++] = ' ';

}

else row->render[index ++] = row-> chars[j];

}

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

row -> rsize = index;

updateSyntax(row);

}

void insertRow(int insertAt, char \*s, size\_t len) {

if ( insertAt < 0 || insertAt > editor.numrows) return;

editor.row = realloc(editor.row, sizeof(editorRow) \* (editor.numrows + 1));

memmove(&editor.row[insertAt + 1], &editor.row[insertAt], sizeof(editorRow) \* (editor.numrows - insertAt));

for (int j = insertAt + 1; j <= editor.numrows; j ++) editor.row[j].index ++;

editor.row[insertAt].index = insertAt;

editor.row[insertAt].size = len;

editor.row[insertAt].chars = malloc(len + 1);

memcpy(editor.row[insertAt].chars, s, len);

editor.row[insertAt].chars[len] = '\0';

editor.row[insertAt].rsize = 0;

editor.row[insertAt].render = NULL;

editor.row[insertAt].hl = NULL;

editor.row[insertAt].hlOpenComment = 0;

updateRow(&editor.row[insertAt]);

editor.numrows ++;

editor.dirty ++;

}

void freeRow(editorRow \*row) {

free(row -> render);

free(row -> chars);

free(row -> hl);

}

void delRow(int at) {

if (at < 0 || at >= editor.numrows) return;

freeRow(&editor.row[at]);

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

for (int j = at + 1; j <= editor.numrows; j ++) editor.row[j].index ++;

editor.numrows --;

editor.dirty ++;

}

void rowInsertChar(editorRow \*row, int insertAt, int c) {

if (insertAt < 0 || insertAt > row -> size) insertAt = row -> size;

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

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

row -> size++;

row -> chars[insertAt] = c;

updateRow(row);

editor.dirty ++;

}

void rowAppendString(editorRow \*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';

updateRow(row);

editor.dirty ++;

}

void rowDelChar(editorRow \*row, int at) {

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

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

row -> size --;

updateRow(row);

editor.dirty ++;

}

/\*\*\* editor operations \*\*\*/

void editorInsertChar(int c) {

if (editor.yCoord == editor.numrows)

insertRow(editor.numrows, "", 0);

rowInsertChar(&editor.row[editor.yCoord], editor.xCoord, c);

editor.xCoord ++;

}

void editorInsertNewline() {

if (editor.xCoord == 0) {

insertRow(editor.yCoord, "", 0);

}

else {

editorRow \* row = &editor.row[editor.yCoord];

insertRow(editor.yCoord + 1, &row -> chars[editor.xCoord], row -> size - editor.xCoord);

row = &editor.row[editor.yCoord];

row -> size = editor.xCoord;

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

updateRow(row);

}

editor.yCoord ++;

editor.xCoord = 0;

}

void editorDelChar() {

if (editor.yCoord == editor.numrows) return;

if (editor.xCoord == 0 && editor.yCoord == 0) return;

editorRow \* row = &editor.row[editor.yCoord];

if (editor.xCoord > 0) {

rowDelChar(row, editor.xCoord - 1);

editor.xCoord --;

}

else {

editor.xCoord = editor.row[editor.yCoord - 1].size;

rowAppendString(&editor.row[editor.yCoord - 1], row -> chars, row -> size);

delRow(editor.yCoord);

editor.yCoord --;

}

}

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

void editorOpen(char \*fileName) {

free(editor.fileName);

editor.fileName = strdup(fileName);

selectSyntaxHighlight();

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

if (!fp) handleError("fopen");

char \*line = NULL;

size\_t lineCapacity = 0;

ssize\_t linelen;

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

while (linelen > 0 && (line[linelen - 1] == '\n' || line[linelen - 1] == '\r'))

linelen--;

insertRow(editor.numrows, line, linelen);

}

free(line);

fclose(fp);

editor.dirty = 0;

}

char \*rowsToString(int \*bufferLen) {

int totalLen = 0;

for (int j = 0; j < editor.numrows; j ++)

totalLen += editor.row[j].size + 1;

\*bufferLen = totalLen;

char \*buf = malloc(totalLen);

char \*p = buf;

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

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

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

\*p = '\n';

p ++;

}

return buf;

}

void editorSave() {

if (editor.fileName == NULL) {

editor.fileName = prompt("\x1b[34mSave as: %s (ESC to cancel)", NULL);

if (editor.fileName == NULL) {

setStatusMessage("\x1b[36m Save aborted\x1b[m");

return;

}

selectSyntaxHighlight();

}

int len;

char \*buf = rowsToString(&len);

int fd = open(editor.fileName, O\_RDWR | O\_CREAT, 0644);

if (fd != -1) {

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

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

close(fd);

free(buf);

editor.dirty = 0;

setStatusMessage("\x1b[32m %d bytes written to disk\x1b[m", len);

return;

}

}

close(fd);

}

free(buf);

setStatusMessage("\x1b[31m Can't save! I/O error: %s\x1b[m", strerror(errno));

}

/\*\*Find\*\*/

void editorFindCallback(char \*sequence, int key) { //for incremental search

static int last\_match = -1;

static int direction = 1;

static int saved\_hl\_line;

static char \*saved\_hl = NULL;

if (saved\_hl) {

memcpy(editor.row[saved\_hl\_line].hl, saved\_hl, editor.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;

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

current += direction;

if ( current == -1) current = editor.numrows - 1;

else if (current == editor.numrows) current = 0;

editorRow \*row = &editor.row[current];

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

if (match) {

last\_match = current;

editor.yCoord = current;

editor.xCoord = rxToxCoord(row, match - row -> render);

editor.rowOffset = editor.numrows;

saved\_hl\_line = current;

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

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

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

break;

}

}

}

void editorFind() {

int saved\_cx = editor.xCoord;

int saved\_cy = editor.yCoord;

int saved\_colOff = editor.colOffset;

int saved\_rowOff = editor.rowOffset;

char \*sequence = prompt("\x1b[32mSearch: %s (Arrows to navigate | Enter to search | ESC to cancel)\x1b[m", editorFindCallback);

if (sequence) free(sequence);

else {

editor.xCoord = saved\_cx;

editor.yCoord = saved\_cy;

editor.rowOffset = saved\_rowOff;

editor.colOffset = saved\_colOff;

}

}

/\*\*\* input \*\*\*/

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

size\_t bufferSize = 128;

char \*buffer = malloc(bufferSize);

size\_t bufferLen = 0;

buffer[0] = '\0';

while (1) {

setStatusMessage(message, buffer);

refreshScreen();

int c = readKey();

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

if (bufferLen != 0) buffer[-- bufferLen] = '\0';

}

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

setStatusMessage("");

if (callback) callback(buffer, c);

free(buffer);

return NULL;

}

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

if ( bufferLen != 0) {

setStatusMessage("");

if (callback) callback(buffer, c);

return buffer;

}

}

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

if ( bufferLen == bufferSize - 1) {

bufferSize \*= 2;

buffer = realloc(buffer, bufferSize);

}

buffer[bufferLen ++] = c;

buffer[bufferLen] = '\0';

}

if (callback) callback(buffer, c);

}

}

void moveCursor(int key) {

editorRow \*row = (editor.yCoord >= editor.numrows) ? NULL : &editor.row[editor.yCoord];

switch (key) {

case ARROW\_LEFT:

if (editor.xCoord != 0)

editor.xCoord --;

else if (editor.yCoord > 0) {

editor.yCoord --;

editor.xCoord = editor.row[editor.yCoord].size;

}

break;

case ARROW\_RIGHT:

if ( row && editor.xCoord < row -> size)

editor.xCoord ++;

else if (row && editor.xCoord == row -> size) {

editor.yCoord ++;

editor.xCoord = 0;

}

break;

case ARROW\_UP:

if (editor.yCoord != 0)

editor.yCoord --;

break;

case ARROW\_DOWN:

if (editor.yCoord < editor.numrows)

editor.yCoord ++;

break;

}

row = (editor.yCoord >= editor.numrows) ? NULL : &editor.row[editor.yCoord];

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

if (editor.xCoord > rowlen) {

editor.xCoord = rowlen;

}

}

void processKey() {

static int quit\_times = 1;

int c = readKey();

switch (c) {

case '\r':

editorInsertNewline();

break;

case CTRL\_KEY('q'):

if (editor.dirty && quit\_times) {

setStatusMessage("\x1b[31m WARNING!! This file contains unsaved changes. Press Ctrl+Q again to exit\x1b[m");

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:

editor.xCoord = 0;

break;

case END\_KEY:

if (editor.yCoord < editor.numrows)

editor.xCoord = editor.row[editor.yCoord].size;

break;

case CTRL\_KEY('f'):

editorFind();

break;

case BACK\_SPACE:

case CTRL\_KEY('h'):

case DEL\_KEY:

if ( c == DEL\_KEY) moveCursor(ARROW\_RIGHT);

editorDelChar();

break;

case PAGE\_UP:

case PAGE\_DOWN:

if (c == PAGE\_UP)

editor.yCoord = editor.rowOffset;

else if (c == PAGE\_DOWN)

editor.yCoord = editor.rowOffset + editor.terminalRows - 1;

if (editor.yCoord > editor.numrows) editor.yCoord = editor.numrows;

int times = editor.terminalRows;

while (times --)

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

break;

case ARROW\_UP:

case ARROW\_DOWN:

case ARROW\_LEFT:

case ARROW\_RIGHT:

moveCursor(c);

break;

case CTRL\_KEY('l'):

case '\x1b':

break;

default :

editorInsertChar(c);

break;

}

quit\_times = 1;

}

/\*\*\* MAIN \*\*\*/

void initEditor() {

editor.xCoord = editor.yCoord = 0;

editor.rx = 0;

editor.rowOffset = editor.colOffset = 0;

editor.numrows = 0;

editor.dirty = 0;

editor.row = NULL;

editor.fileName = NULL;

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

editor.statusmsg\_time = 0;

editor.syntax = NULL;

if (getWindowSize(&editor.terminalRows, &editor.terminalCols) == -1) handleError(" getWindowSize");

editor.terminalRows -= 2; // one for status bar and one for message

} // initializing all the fields of configurations

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

enableRawMode();

initEditor();

if ( argc >= 2) editorOpen(argv[1]);

//editorOpen();

//enabling raw mode to process every character as they're entered

//like entering a password

setStatusMessage("\x1b[34m [Ctrl+Q = quit | Ctrl+S = save | Ctrl+F = find]\x1b[m");

while (1) {

refreshScreen();

processKey();

}

//tcsetattr(STDIN\_FILENO, TCSAFLUSH, &originalTerminal);

//turning raw mode off once we're done

return 0;

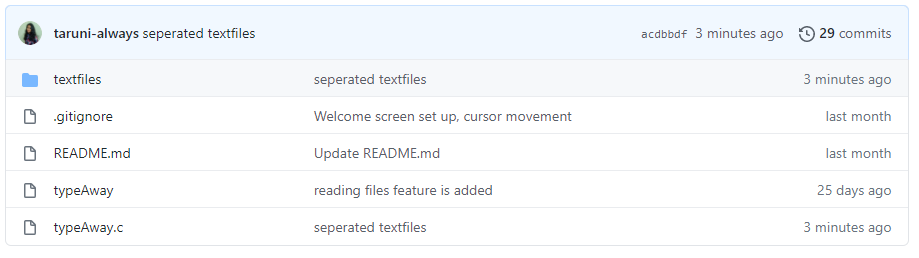
}

### 

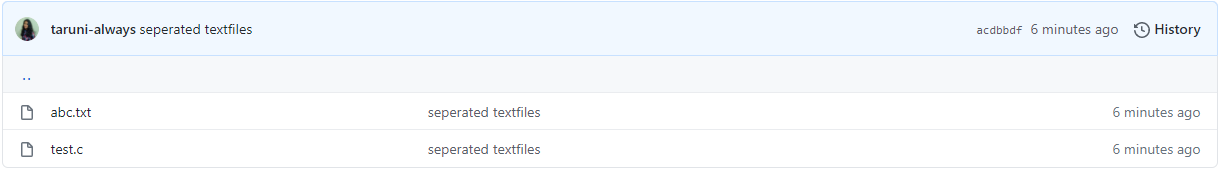
### 3.2.2. GITHUB/FOLDER STRUCTURE

The main code of the text editor is present in the typeAway.c file. The README file has a one-line description of our project. There is a folder named “textfiles” which contains the files we used to test our code.

**GitHub Repo Link:** <https://github.com/taruni-always/typeAway>



Within the “textfiles” folder, we have,



### 

## 

## 3.3. TESTING:

Testing is a method to check whether the actual product matches the expected requirements and to ensure that the product is defect-free. This process involves execution of various parts of the product either using manual or automated tools. The purpose is to identify errors, gaps or missing requirements in contrast to the actual requirements

### USER TEST CASES:

#### 3.3.1. OPEN DOCUMENT

|  |  |  |
| --- | --- | --- |
| Test Case ID: TC01 | | Use Case ID:UC01 |
| Test Case Title: Open Existing Document | |
| Test Case Description: User attempts to open an existing Document. | |
| Test Steps: | Expected Result: | Actual Result: |
| 1. The user gives the name of the document to be opened. | System should display the contents of the document and the user should be able to read/write. | The document mentioned by the user is opened. The user can read/write. |

|  |  |  |
| --- | --- | --- |
| Test Case ID: TC02 | | Use Case ID:UC01 |
| Test Case Title: Open New Document | |
| Test Case Description:User attempts to open a new Document. | |
| Test Steps: | Expected Result: | Actual Result: |
| 1. The user attempts to open a new document by not giving any filename. | System should open a new document for the user to write. | A new document is opened and the user can write into it. |

### 

#### 3.3.2. QUIT DOCUMENT

|  |  |  |
| --- | --- | --- |
| Test Case ID: TC03 | | Use Case ID:UC02 |
| Test Case Title: Quitting before saving | |
| Test Case Description: User attempts to quit from a document without saving the changes made. | |
| Test Steps: | Expected Result: | Actual Result: |
| 1. The user presses the combination “Ctrl+q” to quit. | The system should warn the user that he/she is trying to quit before saving the changes made. | **The System displays “WARNING!! This file contains unsaved changes. Press Ctrl+Q again to exit”.** |

### 

|  |  |  |
| --- | --- | --- |
| Test Case ID: TC04 | | Use Case ID:UC02 |
| Test Case Title:Quitting after saving | |
| Test Case Description: User attempts to quit from a document after saving the changes made. | |
| Test Steps: | Expected Result: | Actual Result: |
| 1.The user presses the combination “Ctrl+q” to quit. | The system should quit from the file. | The document is closed successfully. |

#### 

#### 3.3.3. SAVE DOCUMENT

|  |  |  |
| --- | --- | --- |
| Test Case ID: TC05 | | Use Case ID:UC03 |
| Test Case Title: Saving changes made to an existing document | |
| Test Case Description: User attempts to save the changes made to an existing document | |
| Test Steps: | Expected Result: | Actual Result: |
| 1. The user presses the combination “Ctrl+s” to save the changes. | The changes made should be successfully saved. | The changes made are saved and the system displays “xyz bytes written to disk”, where xyz is the number of bytes written. |

#### 

|  |  |  |
| --- | --- | --- |
| Test Case ID: TC06 | | Use Case ID:UC03 |
| Test Case Title: Saving changes in a new document | |
| Test Case Description: User attempts to save the contents written into a new document | |
| Test Steps: | Expected Result: | Actual Result: |
| The user presses the combination “Ctrl+s” to save the changes. **The user enters the filename for the document to be saved as.** | The system should prompt the user for a name and the changes made should be successfully saved. | The system prompts “Save as:” for the user to enter the filename. The document is saved with the given filename. |

### 

#### 3.3.4. SYNTAX HIGHLIGHTING

|  |  |  |
| --- | --- | --- |
| Test Case ID: TC07 | | Use Case ID:UC04 |
| Test Case Title: Syntax Highlighting | |
| Test Case Description: Syntax Highlighting | |
| Test Steps: | Expected Result: | Actual Result: |
| 1. User opens a document that has been saved at least once. | The system should detect the file type and highlight the necessary syntaxes. | The system detects the file type and highlights the necessary syntaxes. |

#### 

#### 3.3.5. SEARCH

|  |  |  |
| --- | --- | --- |
| Test Case ID: TC08 | | Use Case ID:UC05 |
| Test Case Title: Searching for a word/words | |
| Test Case Description: The user attempts to search for a word in the opened document. | |
| Test Steps: | Expected Result: | Actual Result: |
| 1. The user presses the combination “Ctrl+f”.2. The User types the sequence of characters to find the word. **The user can navigate through multiple occurrences of the word**  **the user presses “Enter” key or “ESC” key.** | The system should point the cursor to the first occurrence of the word, | The system prompts “Search:” for the user to enter the sequence. The user can navigate using arrow keys. When the user presses the “Enter” key, the search prompt is closed and the cursor points to the desired word. If the user presses the “ESC” key instead, he/she comes out of the search but the cursor doesn’t point to the desired word. |

#### 

# 

# 

# 

# 

# 

# 

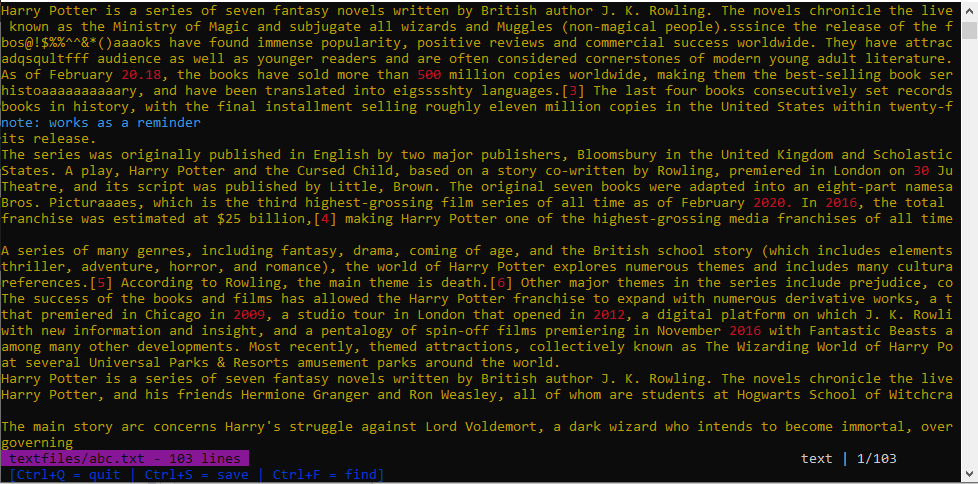
# 4. RESULTS

We were successful in developing a text editor for linux using C language with attractive colour palettes.

## USER TEST CASE RESULTS

### 

### Test Case 1: Open Existing Document



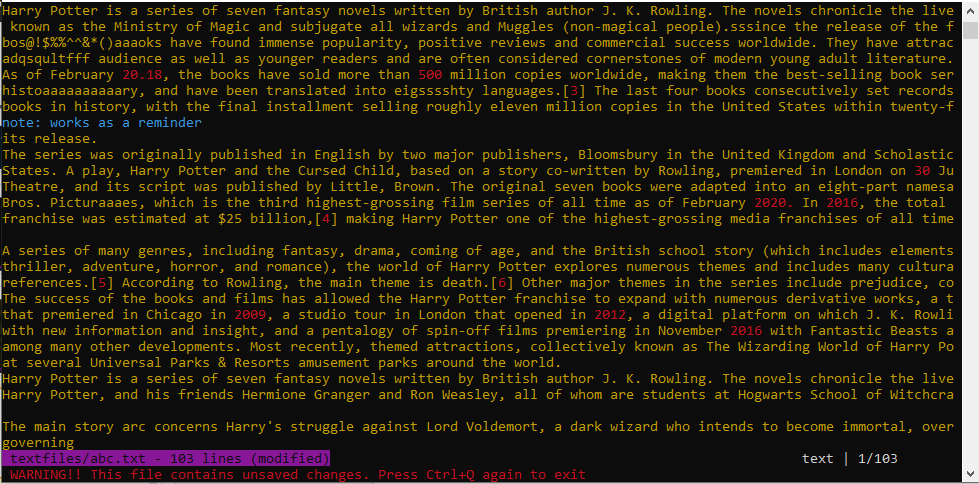
### Test Case 2: Open New Document

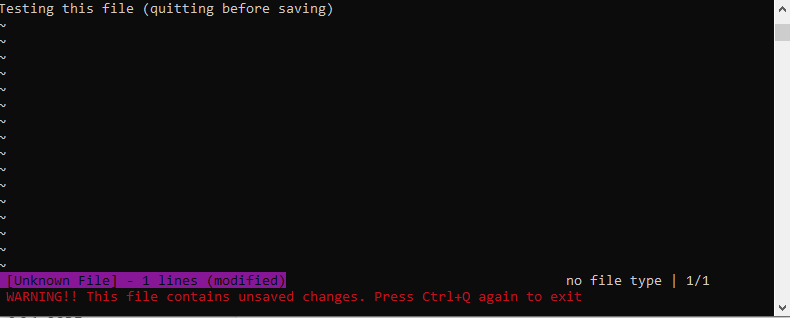
### 

### 

### 

### Test Case 3: Quitting Before Saving





### Test Case 4: Quitting After Saving

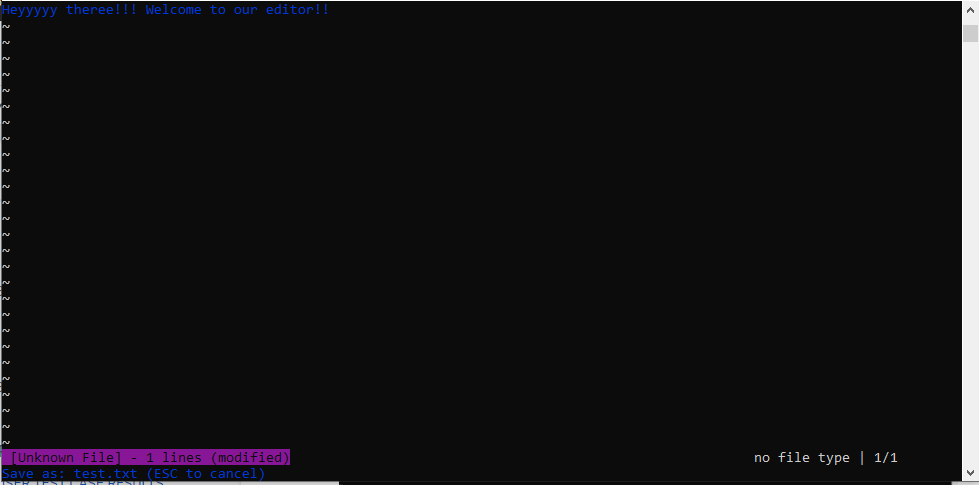
### 

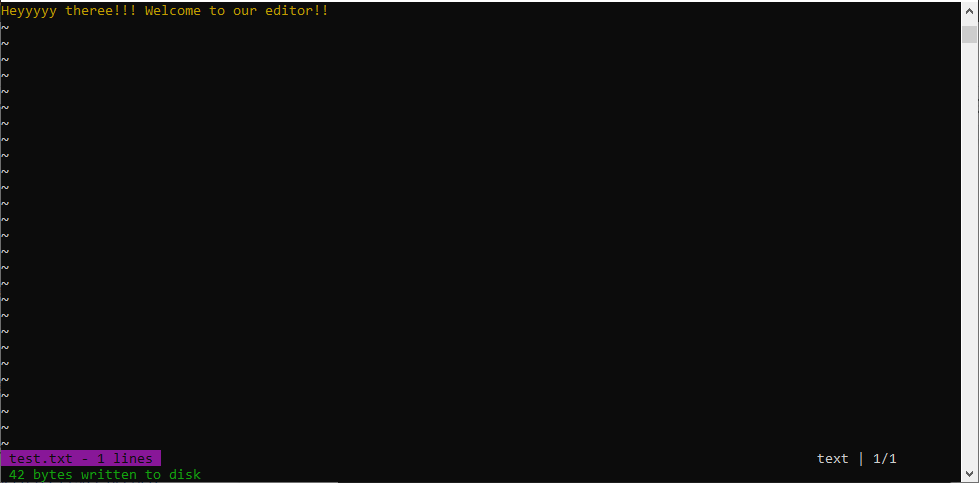
### Test Case 5: Saving Changes Made To An Existing Document

### 

### 

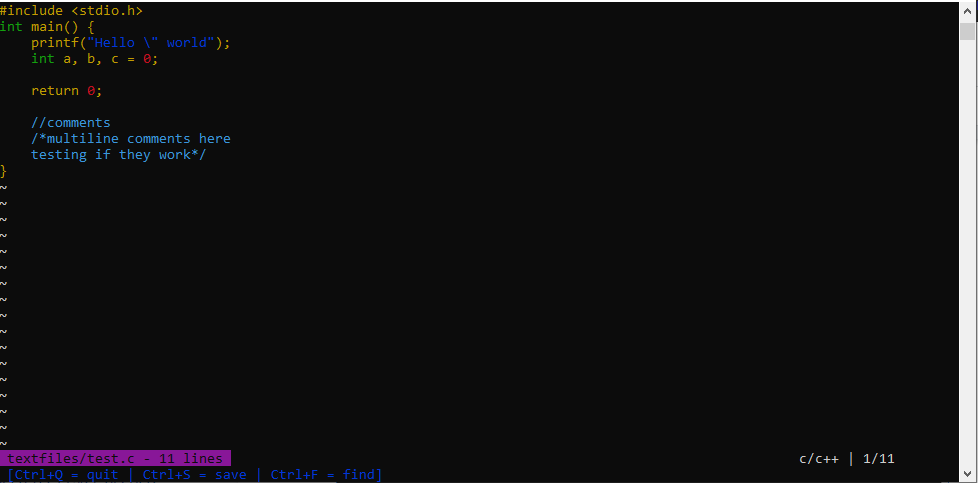
### Test Case 6: Saving Changes In A New Document





### 

### Test Case 7: Syntax Highlighting



### Test Case 8: Searching For A word/words

### 

# 

# 

# 

# 

# 

# 

# 

# 5. ADDITIONAL KNOWLEDGE ACQUIRED

By implementing this project, we are being introduced to many other libraries like ‘termios.h’, ‘time.h’, ‘errno.h’, ‘sys/ioctl.h’, ‘stdarg.h’ and ‘fcntl.h’. We were able to use our knowledge of structures, files, i/o handling and pointers to complete this project. We learnt how to make the terminal work in raw mode to process every character/key pressed.

Apart from this, we learn the value of teamwork, coordination and cooperation. We understand how important these skills are to work towards a common goal and successfully complete the project.

# 

# 6. CONCLUSION AND FUTURE WORK

In conclusion, we have built a text editor for linux, using C language in which users can read or write into new or existing files. The intention behind this project was to enhance our knowledge in these crucial subjects and further provide the same to all users of our platform. We also wanted to inculcate the practise of Self-Learning, that is, without guidance of professors or institutions. Our motive is for students to take ownership of their learning and additionally build independence.

At present, we have only added syntax highlighting for C-type files. In future, we would add more file types like Python, HTML, Java etc. We would also like to add an auto indentation feature in our text editor to improve user comfort. One more thing we would work upon is adding line numbers for any document which improves readability for our users. We would also like to add the feature of copy-paste and cut-paste which lets the user reuse the existing text/code.

# 

# 

# 

# 

# 7. REFERENCES

C Language Documentation (For referring C libraries):

<https://devdocs.io/c/>

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

Stack Overflow (For Debugging Errors):

<https://stackoverflow.com/>

https://www.geeksforgeeks.org/

Reference for Setting Colours in Console:

<https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/color>

<https://bluesock.org/~willkg/dev/ansi.html>

Reference for escape sequences

[https://vt100.net/docs/vt100-ug/chapter3.html](https://vt100.net/docs/vt100-ug/chapter3.html#ED)