Text Editor

21.11.2022

—

Atara Rushi
Roll No: 22
Division: E
Branch: Computer Engineering

Definition

Create a Text Editor With Basic Functionalities.

Functionalities

1. Create File
2. Edit File
3. Open File
4. Save File

Program

```
#define _DEFAULT_SOURCE;
#define _BSD_SOURCE;
#define _GNU_SOURCE;

#include <ctype.h>
#include <errno.h>
#include <fcntl.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <termios.h>
#include <time.h>
#include <unistd.h>

/* Defines */

#define CTRL_KEY(k) ((k) & 0x1f)
#define MEOW_VERSION "0.0.1b"
#define MEOW_TAB_STOP 8
#define MEOW_QUIT_CONFIRM_TIMES 2

enum editor_keys {
```

```
	BACKSPACE = 127,
	ARROW_LEFT = 1000,
	ARROW_RIGHT,
	ARROW_UP,
	ARROW_DOWN,
	DELETE_KEY,
	HOME_KEY,
	END_KEY,
	PAGE_UP,
	PAGE_DOWN
};

/* Data */

typedef struct editor_row {
	int size;
	int rsize;
	char *characters;
	char *render;
} editor_row;

typedef
struct editor_configuration_t {
	int x_coordinate;
	int y_coordinate;
	int rx_coordinate;
	int row_offset;
	int col_offset;
	int screen_rows;
	int screen_cols;
	int number_of_rows;
	int dirty_flag;

	char *file_name;
	char status_message[80];
	time_t status_message_time;
	editor_row *row;

	struct termios original_termios;
} editor_configuration_t;
editor_configuration_t editor_config;

/*
```

```
	Function Prototypes
*/

void editor_set_status_message(const char *fmt, ...);
void editor_refresh_screen();
char *editor_prompt(char *prompt);

/*
	Terminal
*/

void die(const char *s) {
	write(STDOUT_FILENO, "\x1b[2J", 4);
	write(STDOUT_FILENO, "\x1b[H", 3);
	perror(s);
	exit(1);
}

void disable_raw_mode(void) {
	tcsetattr(STDIN_FILENO, TCSAFLUSH, &editor_config.original_termios);
	die("tcsetattr");
	return;
}

void enable_raw_mode(void) {
	if(tcgetattr(STDIN_FILENO, &editor_config.original_termios) == -1)
die("tcgetattr");
	atexit(disable_raw_mode);

	struct termios raw = editor_config.original_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");
	return;
}

int read_key(void) {
	int nread;
```

```
	char c;
	while((nread = read(STDIN_FILENO, &c, 1)) != 1) {
	if(nread == -1 && errno != EAGAIN) die("read");
	}

	if(c == '\x1b') {
	char sequence[3];

	if(read(STDIN_FILENO, &sequence[0], 1) != 1) {
		return '\x1b';
	}
	if(read(STDIN_FILENO, &sequence[1], 1) != 1) {
		return '\x1b';
	}

	if(sequence[0] == '[') {
		if(sequence[1] >= '0' && sequence[1] <= '9') {
			if(read(STDIN_FILENO, &sequence[2], 1) != 1) {
			return '\x1b';
			}
			if(sequence[2] == '~') {
			switch(sequence[1]) {
				case '1': return HOME_KEY;
				case '3': return DELETE_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(sequence[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(sequence[0] == '0') {
		switch(sequence[1]) {
```

```
                case 'H': return HOME_KEY;
                case 'F': return END_KEY;
            }
        }

        return '\x1b';
        } else {
        return c;
        }
}

int get_cursor_position(int *rows, int *cols) {
        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", rows, cols) != 2) {
        return -1;
        }

        return 0;
}

int get_window_size(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 get_cursor_position(rows, cols);
	} else {
	*cols = ws.ws_col;
	*rows = ws.ws_row;
	return 0;
	}
	return -1;
}

/*
	ROW Operations
*/

int editor_row_char_idx_to_render_idx(editor_row *row, int x_coordinate) {
	int rx_coordinate = 0;
	for(int j = 0; j < x_coordinate; j++) {
	if(row -> characters[j] == '\t') {
		rx_coordinate += (MEOW_TAB_STOP - 1) - (rx_coordinate %
MEOW_TAB_STOP);
	}
	rx_coordinate++;
	}
	return rx_coordinate;
}

void editor_update_row(editor_row *row) {
	int tabs = 0;
	for(int j = 0;j < row -> size; j++) {
	if(row -> characters[j] == '\t') {
		tabs++;
	}
	}

	free(row -> render);
	row -> render = malloc(row -> size + tabs * (MEOW_TAB_STOP - 1) + 1);

	int idx = 0;
	for(int j = 0; j < row -> size; j++) {
	if(row -> characters[j] == '\t') {
```

```
            row -> render[idx++] = ' ';
            while(idx % MEOW_TAB_STOP != 0) {
                  row -> render[idx++] = ' ';
            }
      } else {
            row -> render[idx++] = row -> characters[j];
      }

      }

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

      return;
}

void editor_insert_row(int at, char *string, size_t length) {
      if(at < 0 || at > editor_config.number_of_rows) {
      return;
      }
      editor_config.row = realloc(editor_config.row, sizeof(editor_row) *
(editor_config.number_of_rows + 1));
      memmove(&editor_config.row[at + 1], &editor_config.row[at],
sizeof(editor_row) * (editor_config.number_of_rows - at));

      editor_config.row[at].size = length;
      editor_config.row[at].characters = malloc(length + 1);
      memcpy(editor_config.row[at].characters, string, length);
      editor_config.row[at].characters[length] = '\0';

      editor_config.row[at].rsize = 0;
      editor_config.row[at].render = NULL;

      editor_update_row(&editor_config.row[at]);

      editor_config.number_of_rows++;
      editor_config.dirty_flag++;
      return;
}

void editor_free_row(editor_row *row) {
      free(row -> render);
      free(row -> characters);
```

```
	return;
}

void editor_delete_row(int at) {
	if(at < 0 || at >= editor_config.number_of_rows) {
	return;
	}

	editor_free_row(&editor_config.row[at]);
	memmove(&editor_config.row[at], &editor_config.row[at + 1],
sizeof(editor_row) * (editor_config.number_of_rows - at - 1));
	editor_config.number_of_rows--;
	editor_config.dirty_flag++;

	return;
}

void editor_row_insert_char(editor_row *row, int at, int c) {
	if(at < 0 || at > row -> size) {
	at = row -> size;
	}

	row -> characters = realloc(row -> characters, row -> size + 2);
	memmove(&row -> characters[at + 1], &row -> characters[at], row ->
size - at + 1);
	row -> size++;
	row -> characters[at] = c;
	editor_update_row(row);
	editor_config.dirty_flag++;

	return;
}

void editor_row_append_string(editor_row *row, char *string, size_t length)
{
	row -> characters = realloc(row -> characters, row -> size + length +
1);
	memcpy(&row -> characters[row -> size], string, length);
	row -> size += length;
	row -> characters[row -> size] = '\0';
	editor_update_row(row);
	editor_config.dirty_flag++;
```

```
	return;
}

void editor_row_delete_char(editor_row *row, int at) {
	if(at < 0 || at >= row -> size) {
	return;
	}

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

	editor_config.dirty_flag++;

	return;
}

/*
	EDITOR OPERATIONS
*/

void editor_insert_char(int c) {
	if(editor_config.y_coordinate == editor_config.number_of_rows) {
	editor_insert_row(editor_config.number_of_rows, "", 0);
	}

editor_row_insert_char(&editor_config.row[editor_config.y_coordinate],
editor_config.x_coordinate, c);
	editor_config.x_coordinate++;

	return;
}

void editor_insert_new_line(void) {
	if(editor_config.x_coordinate == 0) {
	editor_insert_row(editor_config.y_coordinate, "", 0);
	} else {
	editor_row *row = &editor_config.row[editor_config.y_coordinate];
	editor_insert_row(editor_config.y_coordinate + 1, &row ->
characters[editor_config.x_coordinate], row -> size -
editor_config.x_coordinate);
	row = &editor_config.row[editor_config.y_coordinate];
```

```
	row -> size = editor_config.x_coordinate;
	row -> characters[row -> size] = '\0';
	editor_update_row(row);
	}

	editor_config.y_coordinate++;
	editor_config.x_coordinate = 0;
}

void editor_delete_char(void) {
	if(editor_config.y_coordinate == editor_config.number_of_rows) {
	return;
	}

	if(editor_config.x_coordinate == 0 && editor_config.y_coordinate ==
0) {
	return;
	}

	editor_row *row = &editor_config.row[editor_config.y_coordinate];
	if(editor_config.x_coordinate > 0) {
	editor_row_delete_char(row, editor_config.x_coordinate - 1);
	editor_config.x_coordinate--;
	} else {
	editor_config.x_coordinate =
editor_config.row[editor_config.y_coordinate - 1].size;

editor_row_append_string(&editor_config.row[editor_config.y_coordinate -
1], row -> characters, row -> size);
	editor_delete_row(editor_config.y_coordinate);
	editor_config.y_coordinate--;
	}
	return;
}

/*
	FILE I/O
*/

char *editor_rows_to_string(int *buffer_length) {
	int total_length = 0;
	for(int j = 0; j < editor_config.number_of_rows; j++) {
	total_length += editor_config.row[j].size + 1;
```

```
	}
	*buffer_length = total_length;

	char *buffer = malloc(total_length);
	char *p = buffer;

	for(int j = 0; j < editor_config.number_of_rows; j++) {
	memcpy(p, editor_config.row[j].characters,
editor_config.row[j].size);
	p += editor_config.row[j].size;
	*p = '\n';
	p++;
	}

	return buffer;
}

void editor_open_file(char *file_name) {
	free(editor_config.file_name);
	editor_config.file_name = strdup(file_name);

	FILE *fp = fopen(file_name, "r");
	if(!fp) {
	die("open");
	}

	char *line = NULL;
	size_t line_cap = 0;
	ssize_t line_length;

	while((line_length = getline(&line, &line_cap, fp)) != -1) {
	while(line_length > 0 &&
		    (line[line_length - 1] == '\n' ||
		    line[line_length - 1] == '\r')) {
		line_length--;
	}
	editor_insert_row(editor_config.number_of_rows, line, line_length);
	}

	free(line);
	fclose(fp);

	editor_config.dirty_flag = 0;
```

```
	return;
}

void editor_save_file(void) {
	if(editor_config.file_name == NULL) {
	editor_config.file_name = editor_prompt("Save as: %s (ESC to
Cancel)");
	if(editor_config.file_name == NULL) {
		editor_set_status_message("Save Aborted");
		return;
	}
	}

	int length;
	char *buffer = editor_rows_to_string(&length);

	int fd = open(editor_config.file_name, O_RDWR | O_CREAT, 0644);

	if(fd != -1) {
	if(ftruncate(fd, length) != -1) {
		if(write(fd, buffer, length) == length) {
			close(fd);
			free(buffer);
			editor_config.dirty_flag = 0;
			editor_set_status_message("%d bytes written to disk",
length);
			return;
		}
	}
	close(fd);
	}

	free(buffer);
	editor_set_status_message("Can't save! I/O Error: %s",
strerror(errno));
	return;
}

/*
	Append Buffer
*/
```

```
typedef
struct append_buffer_t {
	char *buffer;
	int length;
} append_buffer_t;


#define APPEND_BUFFER_INIT { NULL, 0 }

void append_buffer_append(append_buffer_t *append_buffer, const char
*string, int length) {
	char *new = realloc(append_buffer -> buffer, append_buffer -> length
+ length);

	if(new == NULL) {
	return;
	}

	memcpy(&new[append_buffer -> length], string, length);
	append_buffer -> buffer = new;
	append_buffer -> length += length;

	return;
}

void append_buffer_free(append_buffer_t *append_buffer) {
	free(append_buffer -> buffer);
	return;
}


/*
	Input
*/

char *editor_prompt(char *prompt) {
	size_t buffer_size = 128;
	char *buffer = malloc(buffer_size);

	size_t buffer_length = 0;
	buffer[0] = '\0';

	while(1) {
```

```
	editor_set_status_message(prompt, buffer);
	editor_refresh_screen();

	int c = read_key();
	if(c == DELETE_KEY || c == CTRL_KEY('h') || c == BACKSPACE) {
		if(buffer_length != 0) {
			buffer[--buffer_length] = '\0';
		}
	} else if(c == '\x1b') {
		editor_set_status_message("");
		free(buffer);
		return NULL;
	} else if(c == '\r') {
		if(buffer_length != 0) {
			editor_set_status_message("");
			return buffer;
		}
	} else if(!iscntrl(c) && c < 128) {
		if(buffer_length == buffer_size - 1) {
			buffer_size *= 2;
			buffer = realloc(buffer, buffer_size);
		}

		buffer[buffer_length++] = c;
		buffer[buffer_length] = '\0';
	}
	}
	return buffer;
}

void move_cursor(int key) {
	editor_row *row = (editor_config.y_coordinate >=
editor_config.number_of_rows) ? NULL :
&editor_config.row[editor_config.y_coordinate];

	switch(key) {
	case ARROW_LEFT:
		if(editor_config.x_coordinate != 0) {
			editor_config.x_coordinate -= 1;
		} else if(editor_config.y_coordinate > 0) {
			editor_config.y_coordinate--;
			editor_config.x_coordinate =
editor_config.row[editor_config.y_coordinate].size;
```

```
            }
            break;
      case ARROW_RIGHT:
            if(row && editor_config.x_coordinate < row -> size) {
                  editor_config.x_coordinate += 1;
            } else if(row && editor_config.x_coordinate == row -> size) {
                  editor_config.y_coordinate++;
                  editor_config.x_coordinate = 0;
            }
            break;
      case ARROW_DOWN:
            if(editor_config.y_coordinate < editor_config.number_of_rows) {
                  editor_config.y_coordinate += 1;
            }
            break;
      case ARROW_UP:
            if(editor_config.y_coordinate != 0) {
                  editor_config.y_coordinate -= 1;
            }
            break;
      }

      row = (editor_config.y_coordinate >= editor_config.number_of_rows) ?
NULL : &editor_config.row[editor_config.y_coordinate];
      int row_length = row ? row -> size : 0;
      if(editor_config.x_coordinate > row_length) {
      editor_config.x_coordinate = row_length;
      }

      return;
}

void process_keypress(void) {
      static int quit_confirm_times = MEOW_QUIT_CONFIRM_TIMES;

      int c = read_key();

      switch(c) {
      case '\r':
            editor_insert_new_line();
            break;

      case CTRL_KEY('q'):
```

```
            if(editor_config.dirty_flag && quit_confirm_times > 0) {
                  editor_set_status_message("WARNING! File has unsaved
changes. ""Press Ctrl-Q %d more times to quit.", quit_confirm_times);
                  quit_confirm_times--;
                  return;
            }
            write(STDOUT_FILENO, "\x1b[2J", 4);
            write(STDOUT_FILENO, "\x1b[H", 3);
            exit(0);
            break;

      case CTRL_KEY('s'):
            editor_save_file();
            break;

      case HOME_KEY:
            editor_config.x_coordinate = 0;
            break;
      case END_KEY:
            if(editor_config.y_coordinate < editor_config.number_of_rows) {
                  editor_config.x_coordinate =
editor_config.row[editor_config.y_coordinate].size;
            }
            break;

      case BACKSPACE:
      case CTRL_KEY('h'):
      case DELETE_KEY:
            if(c == DELETE_KEY) {
                  move_cursor(ARROW_RIGHT);
            }
            editor_delete_char();
            break;

      case PAGE_UP:
      case PAGE_DOWN:
            {
                  if(c == PAGE_UP) {
                  editor_config.y_coordinate = editor_config.row_offset;
                  } else if(c == PAGE_DOWN) {
                  editor_config.y_coordinate = editor_config.row_offset +
editor_config.screen_rows - 1;
                  if(editor_config.y_coordinate >
```

```
editor_config.number_of_rows) {
                        editor_config.y_coordinate =
editor_config.number_of_rows;
                }
                }

                int times = editor_config.screen_rows;
                while(times--) {
                move_cursor(c == PAGE_UP ? ARROW_UP : ARROW_DOWN);
                }
          }
          break;

     case ARROW_LEFT:
     case ARROW_RIGHT:
     case ARROW_DOWN:
     case ARROW_UP:
          move_cursor(c);
          break;

     case CTRL_KEY('l'):
     case '\x1b':
          break;

     default:
          editor_insert_char(c);
          break;
     }

     quit_confirm_times = MEOW_QUIT_CONFIRM_TIMES;

     return;
}

/*
     Output
*/

void editor_scroll(void) {
     editor_config.rx_coordinate = 0;
     if(editor_config.y_coordinate < editor_config.number_of_rows) {
     editor_config.rx_coordinate =
editor_row_char_idx_to_render_idx(&editor_config.row[editor_config.y_coordi
```

```
nate], editor_config.x_coordinate);
      }


      if(editor_config.y_coordinate < editor_config.row_offset) {
      editor_config.row_offset = editor_config.y_coordinate;
      }

      if(editor_config.y_coordinate >= editor_config.row_offset +
editor_config.screen_rows) {
      editor_config.row_offset = editor_config.y_coordinate -
editor_config.screen_rows + 1;
      }

      if(editor_config.rx_coordinate < editor_config.col_offset) {
      editor_config.col_offset = editor_config.rx_coordinate;
      }
      if(editor_config.rx_coordinate >= editor_config.col_offset +
editor_config.screen_cols) {
      editor_config.col_offset = editor_config.rx_coordinate -
editor_config.screen_cols + 1;
      }
}

void editor_draw_rows(append_buffer_t *append_buffer) {
      for(int y = 0; y < editor_config.screen_rows; y++) {
      int file_row = y + editor_config.row_offset;
      if(file_row >= editor_config.number_of_rows) {
            if(editor_config.number_of_rows == 0 && y ==
editor_config.screen_rows / 3) {
                  char welcome_message[128];
                  int welcome_message_length = snprintf(welcome_message,
sizeof(welcome_message), "Meow Text Editor -- version %s", MEOW_VERSION);
                  if(welcome_message_length > editor_config.screen_cols) {
                  welcome_message_length = editor_config.screen_cols;
                  }

                  int padding = (editor_config.screen_cols -
welcome_message_length) / 2;
                  if(padding) {
                  append_buffer_append(append_buffer, "~", 1);
                  padding--;
                  }
```

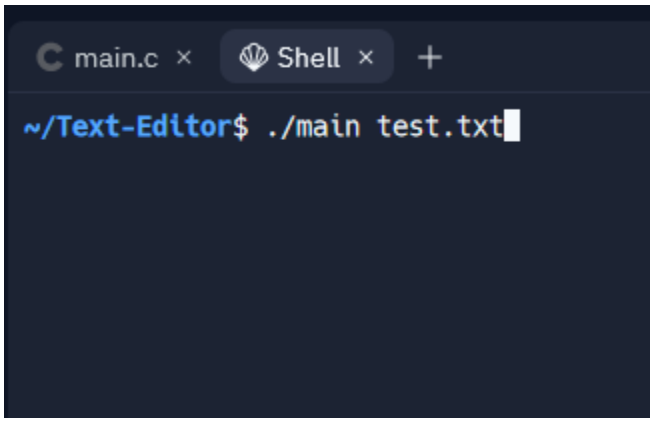
```
                while(padding--) {
                append_buffer_append(append_buffer, " ", 1);
                }

                append_buffer_append(append_buffer, welcome_message,
welcome_message_length);
            } else {
                append_buffer_append(append_buffer, "~", 1);
            }
      } else {
            int length = editor_config.row[file_row].rsize -
editor_config.col_offset;

            if(length < 0) {
                length = 0;
            }
            if(length > editor_config.screen_cols) {
                length = editor_config.screen_cols;
            }

            append_buffer_append(append_buffer,
&editor_config.row[file_row].render[editor_config.col_offset], length);
      }

      append_buffer_append(append_buffer, "\x1b[K", 3);
      append_buffer_append(append_buffer, "\r\n", 2);
      }

      return;
}

void editor_draw_status_bar(append_buffer_t *append_buffer) {
      append_buffer_append(append_buffer, "\x1b[7m", 4);
      char status[80];
      char r_status[80];
      int length = snprintf(status, sizeof(status), "%.20s - %d lines %s",
editor_config.file_name ? editor_config.file_name : "[No Name]",
editor_config.number_of_rows, editor_config.dirty_flag ? "(modified)" :
"");

      int r_length = snprintf(r_status, sizeof(r_status), "%d/%d",
editor_config.y_coordinate + 1, editor_config.number_of_rows);
```

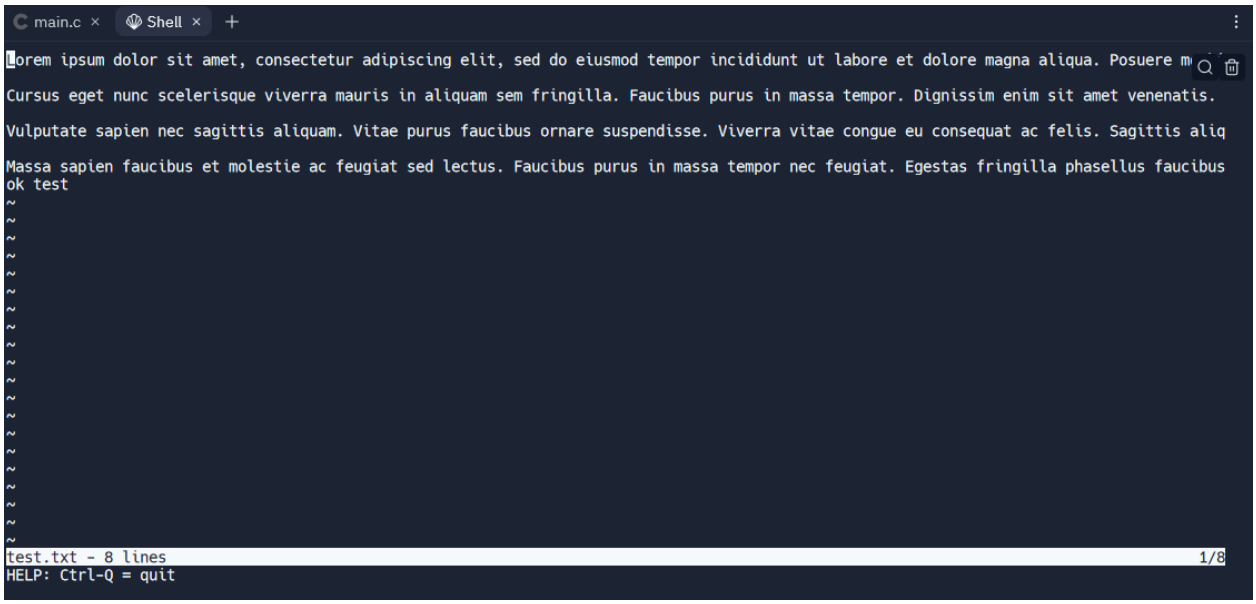
```
	if(length > editor_config.screen_cols) {
	length = editor_config.screen_cols;
	}

	append_buffer_append(append_buffer, status, length);

	while(length < editor_config.screen_cols) {
	if(editor_config.screen_cols - length == r_length) {
		append_buffer_append(append_buffer, r_status, r_length);
		break;
	} else {
		append_buffer_append(append_buffer, " ", 1);
		length++;
	}
	}
	append_buffer_append(append_buffer, "\x1b[m", 3);
	append_buffer_append(append_buffer, "\r\n", 2);

	return;
}

void editor_draw_message_bar(append_buffer_t *append_buffer) {
	append_buffer_append(append_buffer, "\x1b[K", 3);

	int message_length = strlen(editor_config.status_message);
	if(message_length > editor_config.screen_cols) {
	message_length = editor_config.screen_cols;
	}

	if(message_length && time(NULL) - editor_config.status_message_time <
5) {
	append_buffer_append(append_buffer, editor_config.status_message,
message_length);
	}

	return;
}

void editor_refresh_screen(void) {
	editor_scroll();

	append_buffer_t append_buffer = APPEND_BUFFER_INIT;
```

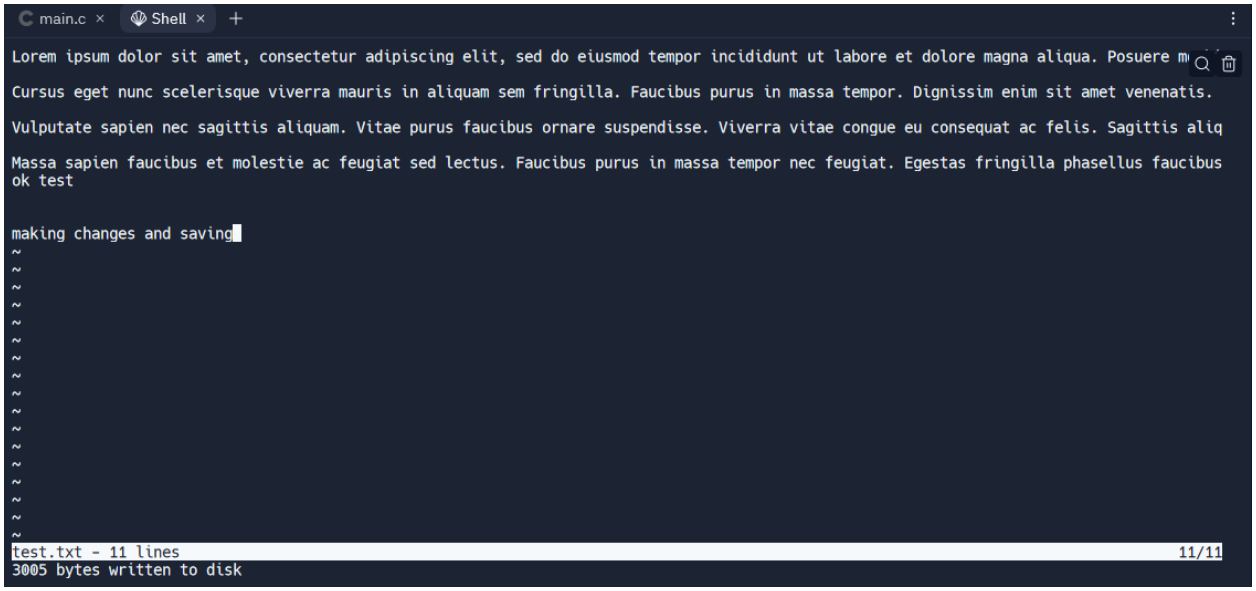
```
	append_buffer_append(&append_buffer, "\x1b[?25l", 6);
	append_buffer_append(&append_buffer, "\x1b[H", 3);

	editor_draw_rows(&append_buffer);
	editor_draw_status_bar(&append_buffer);
	editor_draw_message_bar(&append_buffer);

	char buffer[32];
	snprintf(buffer, sizeof(buffer), "\x1b[%d;%dH",
(editor_config.y_coordinate - editor_config.row_offset) + 1,
(editor_config.rx_coordinate - editor_config.col_offset) + 1);
	append_buffer_append(&append_buffer, buffer, strlen(buffer));

	append_buffer_append(&append_buffer, "\x1b[?25h", 6);

	write(STDOUT_FILENO, append_buffer.buffer, append_buffer.length);

	append_buffer_free(&append_buffer);

	return;
}

void editor_set_status_message(const char *fmt, ...) {
	va_list ap;

	va_start(ap, fmt);
	vsnprintf(editor_config.status_message,
sizeof(editor_config.status_message), fmt, ap);
	va_end(ap);

	editor_config.status_message_time = time(NULL);

	return;
}


/*
	Init
*/

void initialize_editor(void) {
	editor_config.x_coordinate = 0;
	editor_config.y_coordinate = 0;
```

```
	editor_config.rx_coordinate = 0;
	editor_config.row_offset = 0;
	editor_config.col_offset = 0;
	editor_config.number_of_rows = 0;
	editor_config.dirty_flag = 0;
	editor_config.row = NULL;
	editor_config.file_name = NULL;
	editor_config.status_message[0] = '\0';
	editor_config.status_message_time = 0;

	if(get_window_size(&editor_config.screen_rows,
&editor_config.screen_cols) == -1) {
	die("get_window_size");
	}
	editor_config.screen_rows -= 2;
}

int main(int argc, char *argv[]) {
	enable_raw_mode();
	initialize_editor();
	if(argc >= 2) {
	editor_open_file(argv[1]);
	}

	editor_set_status_message("HELP: Ctrl-Q = quit");

	while(1) {
	editor_refresh_screen();
	process_keypress();
	}
	return 0;
}
```

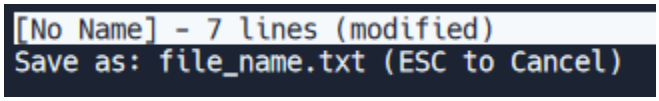
Output

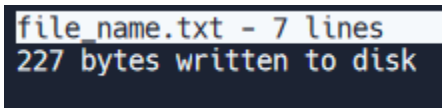

main.c
Shell
~/Text-Editor$ ./main test.txt

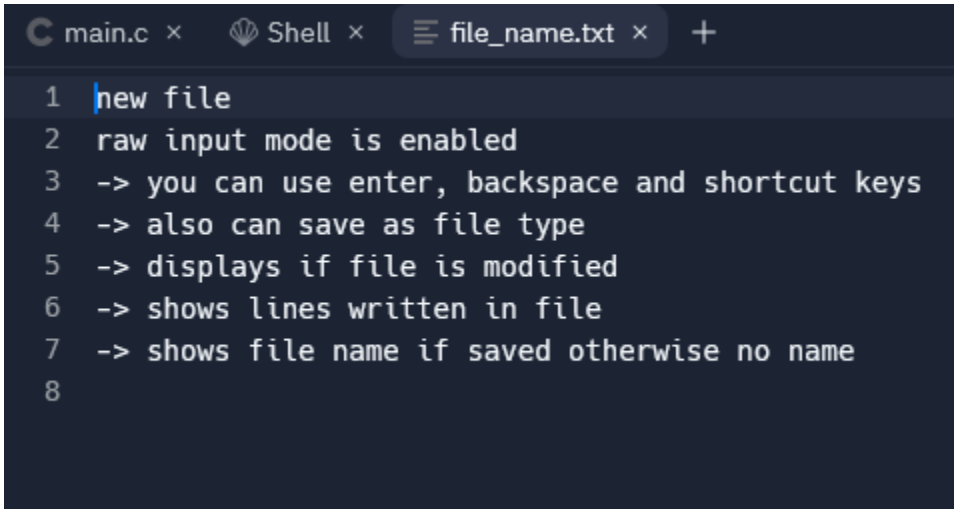

main.c
Shell
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Posuere m
Cursus eget nunc scelerisque viverra mauris in aliquam sem fringilla. Faucibus purus in massa tempor. Dignissim enim sit amet venenatis.
Vulputate sapien nec sagittis aliquam. Vitae purus faucibus ornare suspendisse. Viverra vitae congue eu consequat ac felis. Sagittis aliq
Massa sapien faucibus et molestie ac feugiat sed lectus. Faucibus purus in massa tempor nec feugiat. Egestas fringilla phasellus faucibus
ok test
test.txt - 8 lines
1/8
HELP: Ctrl-Q = quit


main.c
Shell
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Posuere m
Cursus eget nunc scelerisque viverra mauris in aliquam sem fringilla. Faucibus purus in massa tempor. Dignissim enim sit amet venenatis.
Vulputate sapien nec sagittis aliquam. Vitae purus faucibus ornare suspendisse. Viverra vitae congue eu consequat ac felis. Sagittis aliq
Massa sapien faucibus et molestie ac feugiat sed lectus. Faucibus purus in massa tempor nec feugiat. Egestas fringilla phasellus faucibus
ok test
making changes and saving
test.txt - 11 lines
11/11
3005 bytes written to disk

main.c
Shell
new file
raw input mode is enabled
-> you can use enter, backspace and shortcut keys
-> also can save as file type
-> displays if file is modified
-> shows lines written in file
-> shows file name if saved otherwise no name
-> has user friendly status bar
[No Name] - 8 lines (modified)
8/8


[No Name] - 7 lines (modified)
Save as: file_name.txt (ESC to Cancel)


file_name.txt - 7 lines
227 bytes written to disk


main.c
Shell
file_name.txt
1 new file
2 raw input mode is enabled
3 -> you can use enter, backspace and shortcut keys
4 -> also can save as file type
5 -> displays if file is modified
6 -> shows lines written in file
7 -> shows file name if saved otherwise no name
8