**DLITHE PROJECT REPORT**

**PROJECT ID :** CP042

**PROJECT TITLE :** TEXT EDITOR

**TEAM MEMBERS :**

* Uttam(4MT21CS175)
* Vipul(4MT21CS182)
* Srivasta(4MT21CS163)
* Shreys(4MT21CS153)
* Narendra(4MT21CS177)

**REPORT**

**Abstract :**

Text editors are fundamental tools in modern computing, enabling users to create, edit, and manage textual content efficiently. This abstract explores the significance of text editors, their key features, and their diverse applications across various domains.

**Introduction :**

A text editor is more than just a software application; it's a gateway to creativity, productivity, and innovation. Whether you're a writer crafting a novel, a programmer coding the next groundbreaking software, a student taking notes in class, or an administrator managing configuration files on a server, the text editor is your trusted companion.

**Background :**

The history of text editors is a testament to the evolving landscape of computing and the relentless pursuit of efficient tools for manipulating text-based data. In the early days of computing, during the 1950s and 1960s, users grappled with rudimentary line editors. These editors operated in a character-by-character fashion and were often tedious and time-consuming to use.

As personal computers emerged in the 1970s, the text editing landscape began to shift. Richard Stallman's Emacs introduced full-screen text editing, revolutionizing the way users interacted with text. Emacs was notable for its extensibility, providing a platform for users to customize and enhance their text editing experience. Around the same time, Bill Joy created Vi, which introduced a modal editing system and remains a favorite among programmers for its efficiency and keyboard-centric approach.

**Objectives :**

The main objectives of the Text Editor are as follows:

* Efficiency and Productivity
* Usability and Accessibility
* Text Manuplation and Formatting
* Customization and Extensibility
* Cross-Platform Compatilibility
* User-Friendly Interface

**Technologies Used :**

The Text Editor is implemented using the following technologies:

* C programming language
* File handling for data storage

**System Architecture :**

1. User Interface (UI):

Editor Interface: This is where users interact with the text and perform editing tasks. It includes the text display area, input methods, and user interface elements like menus, toolbars, and status bars.

Customization: Options for customizing the user interface, such as themes, fonts, and keybindings, may be provided.

2. Text Editing Engine:

Core Logic: The core of the text editor contains the logic for text manipulation, cursor movement, selection, and other essential editing functions.

Syntax Highlighting: If the text editor supports syntax highlighting, this component is responsible for identifying and highlighting code syntax based on the language being edited.

Undo/Redo History: Maintains a history of user actions for undo and redo functionality.

3. File I/O and Buffer Management:

File Input/Output: This module handles reading from and writing to text files on disk. It manages the open files and ensures data consistency.

Buffer Management: Buffers hold the text content in memory, enabling fast and efficient text manipulation. Buffers may be associated with open files.

4. Search and Replace:

Search Engine: Responsible for searching for text within the document or across multiple documents.

Replace Engine: Manages search and replace operations, enabling users to find and replace text content.

5. Settings and Preferences:

Users can customize various aspects of the text editor's behavior and appearance through settings and preferences.

6. Platform Integration:

- Platform-specific code is included to handle interactions with the underlying operating system, including window management, file dialogs, and clipboard integration.

The above architecture is a generalized representation. In practice, the architecture can become more complex, especially in feature-rich text editors. Web-based text editors may have a different architecture, relying on web technologies and server-client interactions. Mobile text editors may also adapt their architectures for the specific mobile platform and device constraints.

**File Handling in this Program:**

* The program uses file handling to store and retrieve data for users.
* Each entity has its own separate file for data storage.
* File operations such as opening, reading, writing, modifying, and deleting records are performed using file pointers and functions.

**Project Features:**

The project consists of the following features:

1. Create a File

Users can create a new text file by providing a filename and entering the content they want to save in the file.

2. Read a File

The application allows users to read the content of an existing text file. Users can specify the filename, and the program will display the text content on the screen.

3. Update a File

Notepad 2.0 enables users to update the content of an existing text file. Users can specify the filename and enter the new content they want to replace the original content with.

4. Delete a File

Users can delete an existing text file by providing the filename. The application will prompt for confirmation before proceeding with the deletion.

5. Copy, Cut, and Paste

Notepad 2.0 supports basic text manipulation operations. Users can copy, cut, and paste text from one part of the file to another. The application uses familiar keyboard shortcuts such as Ctrl+C (copy), Ctrl+X (cut), and Ctrl+V (paste) for these operations.

6. Find and Replace

Users can search for specific words or phrases within the text file using the "Find" feature. Additionally, they can replace all occurrences of a word or phrase with another provided text using the "Replace" feature.

**Design and Implementation :**

Front-End Design :

Front-end design for a text editor includes an intuitive user interface with a text display area, menus, and toolbars. It should prioritize readability and customization, allowing users to adjust themes, fonts, and layouts. Usability features like syntax highlighting and keyboard shortcuts enhance the editing experience.

Back-End Design :

The back-end design of a text editor encompasses file I/O management, text buffer handling, and core editing logic. It manages text input and output, file interactions, and undo/redo functionality. Additionally, it may support plugins and extensions, syntax highlighting, and version control integration to enhance the editor's capabilities.

**Testing :**

Unit Testing :

Unit testing for a text editor involves testing individual components, like syntax highlighting or file I/O, in isolation to ensure they function correctly and don't introduce errors in the editor.

Integration Testing :

Integration testing for a text editor verifies how different components interact, ensuring that features like file loading, undo/redo, and plugins work seamlessly together to provide a functional and stable editor.

User Acceptance Testing :

User acceptance testing (UAT) for a text editor involves end-users validating the software to ensure it meets their requirements, expectations, and usability standards, ensuring its suitability for their needs.

**Challenges Faced :**

* Cross-platform Compatibility
* Performance Optimization
* User Interface Design
* Syntax Highlighting
* Scalability
* Undo/Redo Functionality

**Appendices:**

Appendix A: Source Code Sample

Provide a sample of the source code highlighting a specific feature or module, such as file I/O, text manipulation, or the user interface.

Appendix B: Build and Installation Guide

Offer step-by-step instructions on how to build and install the text editor on different platforms.

Appendix C: User Manual

Include a user manual that explains how to use the text editor's features, keyboard shortcuts, customization options, and any advanced functionalities.

Appendix D: Syntax Highlighting Guide

If the text editor supports syntax highlighting, document how to define and add syntax highlighting rules for various programming languages.

Appendix E: Data Structures and Algorithms

Explain the data structures and algorithms used in the text editor's core functionality, such as text buffer management, undo/redo, and search algorithms.

Appendix F: Error Handling and Debugging

Detail error-handling strategies, debugging techniques, and common issues that developers may encounter during development.

Appendix G: Performance Optimization

Discuss strategies for optimizing the text editor's performance, including memory management and algorithmic improvements.

Appendix H: Version Control Integration

If the text editor integrates with version control systems, provide instructions on configuring and using these features.

Appendix I: References

List references, books, websites, or research papers that were consulted during the development of the text editor.

**Code Snippets :**

// Notepad 2.0

// Notepad 2.0

#include <stdio.h>

#include <stdlib.h>

#include "functions.h"

int main()

{

FILE \*fptr;

char in, file\_name[35];

int choice;

char \*old\_str[100], \*new\_str[100];

char search\_str[100];

printf("\n\t\t\tNotepad 2.0\n");

do

{

printf("\n\nEnter your choice\n1.Create a file\n2.Read a file\n3.Update a file\n4.Delete a file\n5.Copy and paste a file\n6.Cut and Paste a file\n7.Replace a String\n8.Find\n9.Exit\n");

scanf("%d", &choice);

printf("\n\t\t\tEnter file name: ");

scanf("%s", file\_name);

switch (choice)

{

case 1:

// Write mode

CREATE\_FILE(file\_name);

break;

case 2:

// Read mode

READ\_FILE(file\_name);

break;

case 3:

// Update mode

UPDATE\_FILE(file\_name);

break;

case 4:

// delete the file

DELETE\_FILE(file\_name);

break;

case 5:

// copy a file

COPY\_FILE(file\_name);

break;

case 6:

CUT\_FILE(file\_name);

break;

case 7:

// replace

printf("Enter old substring: ");

scanf("%s", old\_str);

printf("Enter new substring: ");

scanf("%s", new\_str);

REPLACE\_FILE(file\_name, old\_str, new\_str);

break;

case 8:

printf("Enter the string to search for: ");

scanf("%s", search\_str);

// FIND\_FILE(file\_name);

if (isStringInFile(file\_name, search\_str))

{

printf("String found in the file.\n");

}

else

{

printf("String not found in the file.\n");

}

break;

case 9:

// Exit

exit(0);

default:

printf("Invalid choice! try again\n");

break;

}

} while (1);

return 0;

}

// functions.h

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

bool endsWithEnd(const char \*str)

{

size\_t len = strlen(str);

if (len < 3)

{

return false; // String is too short to match "End"

}

return strcmp(str + len - 4, "Stop") == 0;

}

// Read a file

void READ\_FILE(const char \*file\_name)

{

FILE \*fptr;

char in;

fptr = fopen(file\_name, "r");

if (fptr == NULL)

{

printf("File not found or unable to open.\n");

exit(1);

}

printf("\n\n\t\t\tREAD MODE\n");

while ((in = getc(fptr)) != EOF)

{

printf("%c", in);

}

fclose(fptr);

}

// Create a file

void CREATE\_FILE(const char \*file\_name)

{

printf("Press Enter and type Stop to end input.\n");

FILE \*fptr;

bool status;

char content[1000];

fptr = fopen(file\_name, "w");

if (fptr == NULL)

{

printf("File not found or unable to open.\n");

}

printf("\n\t\t\tWRITE MODE\n");

printf("Enter the content to write (Press Ctrl+Z or Ctrl+D to stop):\n");

while (fgets(content, sizeof(content), stdin) != NULL)

{

// Remove the trailing newline character

int len = strlen(content);

if (len > 0 && content[len - 1] == '\n')

{

content[len - 1] = '\0'; // Replace '\n' with '\0' (null character)

}

status = endsWithEnd(content);

if (status)

{

break; // Stop if user enters "End"

}

fputs(content, fptr);

}

fclose(fptr);

}

// Update a file

void UPDATE\_FILE(const char \*file\_name)

{

FILE \*fptr;

bool status;

char content[1000];

fptr = fopen(file\_name, "a");

if (fptr == NULL)

{

printf("File not found or unable to open.\n");

exit(1);

}

printf("\n\n\t\t\tAPPEND MODE\n");

printf("Enter the content to append (Press Ctrl+Z or Ctrl+D to stop):\n");

while (fgets(content, sizeof(content), stdin) != NULL)

{

// Remove the trailing newline character

int len = strlen(content);

if (len > 0 && content[len - 1] == '\n')

{

content[len - 1] = '\0'; // Replace '\n' with '\0' (null character)

}

status = endsWithEnd(content);

if (status)

{

break; // Stop if user enters "End"

}

fputs(content, fptr);

}

fclose(fptr);

}

// Delete a file

void DELETE\_FILE(const char \*fileName)

{

int status = remove(fileName);

if (status == 0)

{

printf("File deleted successfully.\n");

}

else

{

printf("File not found or unable to delete.\n");

}

}

// Copy and Paste a file

void COPY\_FILE(const char \*source\_file)

{

char destination\_file[35];

printf("\n\t\t\tEnter destination file name: ");

scanf("%s", destination\_file);

FILE \*source, \*destination;

char ch;

source = fopen(source\_file, "r");

if (source == NULL)

{

printf("Source file not found or unable to open.\n");

return;

}

destination = fopen(destination\_file, "w");

if (destination == NULL)

{

printf("Unable to create or open the destination file.\n");

fclose(source);

return;

}

while ((ch = fgetc(source)) != EOF)

{

fputc(ch, destination);

}

fclose(source);

fclose(destination);

printf("File copied successfully.\n");

}

// Cut and Paste a file

void CUT\_FILE(const char \*source\_file)

{

char destination\_file[35];

printf("\n\t\t\tEnter destination file name: ");

scanf("%s", destination\_file);

FILE \*source, \*destination;

char buffer[1024];

source = fopen(source\_file, "r");

if (source == NULL)

{

printf("Source file not found or unable to open.\n");

return;

}

destination = fopen(destination\_file, "w");

if (destination == NULL)

{

printf("Unable to create or open the destination file.\n");

fclose(source);

return;

}

while (fgets(buffer, sizeof(buffer), source) != NULL)

{

fputs(buffer, destination);

}

fclose(source);

source = fopen(source\_file, "w");

fclose(source);

fclose(destination);

printf("File copied successfully.\n");

}

// find

// void FIND\_FILE(const char \*file\_name)

// {

// char sub\_str;

// printf("Enter the string to find: ");

// scanf("%s",sub\_str);

// FILE \*fptr;

// char buffer[1024];

// char \*pos;

// int index = -1;

// fptr = fopen(file\_name, "r");

// if (fptr == NULL)

// {

// printf("File not found or unable to open.\n");

// }

// while (fgets(buffer, sizeof(buffer), fptr) != NULL)

// {

// pos = strstr(buffer, sub\_str);

// if (pos != NULL)

// {

// index = pos - buffer;

// break;

// }

// }

// printf("The string is found!\n");

// fclose(fptr);

// }

// replace function

void REPLACE\_FILE(const char \*file\_name, const char \*old\_str, const char \*new\_str) {

FILE \*fptr;

FILE \*tempFile;

char buffer[1024];

char new\_buffer[1024];

char \*pos;

fptr = fopen(file\_name, "r");

if (fptr == NULL) {

printf("File not found or unable to open.\n");

return;

}

tempFile = fopen("temp.txt", "w");

if (tempFile == NULL) {

printf("Error creating temporary file.\n");

fclose(fptr);

return;

}

while (fgets(buffer, sizeof(buffer), fptr) != NULL) {

pos = strstr(buffer, old\_str);

if (pos != NULL) {

// Copy characters before the old\_str

strncpy(new\_buffer, buffer, pos - buffer);

new\_buffer[pos - buffer] = '\0'; // Null-terminate the new string

// Append the new\_str

strcat(new\_buffer, new\_str);

// Append the characters after the old\_str

strcat(new\_buffer, pos + strlen(old\_str));

// Write the modified line to the temporary file

fputs(new\_buffer, tempFile);

} else {

// If old\_str not found, write the original line

fputs(buffer, tempFile);

}

}

fclose(fptr);

fclose(tempFile);

// Remove the original file and rename the temporary file

remove(file\_name);

rename("temp.txt", file\_name);

}

int isStringInFile(const char \*file\_name, const char \*search\_str) {

FILE \*file = fopen(file\_name, "r");

if (file == NULL) {

printf("File not found or unable to open.\n");

return 0; // Return 0 to indicate failure

}

char buffer[1024];

while (fgets(buffer, sizeof(buffer), file) != NULL) {

if (strstr(buffer, search\_str) != NULL) {

fclose(file);

return 1; // Return 1 to indicate success (string found)

}

}

fclose(file);

return 0; // Return 0 to indicate failure (string not found)

}

**Conclusion :**

In conclusion, text editors are indispensable tools in the digital world, serving writers, programmers, and professionals. Their evolution has brought about versatile, customizable, and efficient platforms that empower users to create, edit, and manage textual content effectively and creatively.

**References :**

<https://www.google.com/amp/s/www.geeksforgeeks.org/basics-file-handling-c/amp/>

<https://www.programiz.com/c-programming/c-file-input-output>