A red and black text on a white background

AI-generated content may be incorrect.

**UNIVERSITY INSTITUTE OF COMPUTING**

**PROJECT REPORT ON TEXT EDITOR**

**Program Name: BCA Subject Name**

**Code: Data Structures(24CAT-201)**

**Submitted by: Submitted to:**

**Name:CHRIST Name:**

**Name: Designation:**

**UID:**

**UID:**

**Section:**

**Group:**

Abstract

This report outlines the development of a simple text editor implemented in C. The program allows users to interact with a buffer to add, delete, and display lines of text. The text editor operates in a command-line environment, offering basic editing features through a simple menu-driven interface. The report provides an overview of the program's design, its functionality, and areas for potential improvement**.**

**1. Introduction**

Text editors are essential tools for managing textual information. In many cases, users need to modify, organize, and display textual data. While modern text editors often offer sophisticated features, creating a simple text editor from scratch in C provides a solid understanding of file handling, memory management, and basic user input/output. This report describes a command-line text editor written in C that allows users to add lines, delete lines, and display the contents of the editor.

**2. Program Overview**

The program simulates a basic text editor where the user can:

1. **Add Lines**: Users can input new lines of text.
2. **Delete the Last Line**: Users can remove the most recently added line.
3. **Display the Text**: Users can view all lines currently stored in the buffer.
4. **Exit the Editor**: Users can exit the program.

Here is a small part of the program that demonstrates the functionality of adding a new line to the text:

// Function to add a line

void addLine(const char \*line) {

if (lineCount >= MAX\_LINES) {

printf("Text buffer is full! Cannot add more lines.\n");

return;

}

strncpy(text[lineCount], line, MAX\_LENGTH - 1);

text[lineCount][MAX\_LENGTH - 1] = '\0'; // Ensure null-termination

lineCount++;

printf("Line added.\n");

}

This function is responsible for adding a new line to the text array. It checks whether the buffer has reached its maximum size (MAX\_LINES), and if not, it copies the input line into the array. It also ensures the string is null-terminated to prevent overflow.

**3. Functionality and Features**

A) **Adding a Line (add Line):**

* The function accepts a line of text as input and stores it in the text buffer.
* The input is truncated to ensure that it does not exceed the maximum allowed length (MAX\_LENGTH - 1).
* A counter (lineCount) keeps track of the number of lines stored.

B) **Deleting a Line (delete Line):**

* The function removes the most recent line by decrementing the lineCount.
* If there are no lines in the buffer, the program alerts the user that no lines can be deleted.

C) **Displaying Text (displayText):**

* The function prints the current content of the text buffer, showing all the lines, the user has added.
* If the buffer is empty, the function displays a message indicating that there is no content to display.

D) **Menu System:**

* The program continuously displays a menu prompting the user to choose between adding a line, deleting the last line, displaying the text, or exiting the program.
* Input is taken using scanf () for numeric choices and fgets () for entering the text lines, ensuring that user input is processed safely.

**4. Code Structure**

* **Global Variables**:
  + text [MAX\_LINES] [MAX\_LENGTH]: A 2D array to hold the lines of text.
  + lineCount: An integer to track the number of lines currently stored.
* **Fonctions**:
  + add Line (const char \*line): Adds a new line to the text buffer.
  + delete Line (): Deletes the most recent line.
  + display Text (): Displays the content of the text buffer.
  + main (): Manages the user interface, presenting the menu, and handling user input.

**5. Error Handling**

* **Buffer Full Check**: The add Line () function checks if the buffer has reached its maximum capacity (MAX\_LINES). If the buffer is full, it prevents further lines from being added and informs the user.
* **Empty Buffer Check**: The delete Line () and display Text () functions check whether any lines exist in the buffer. If the buffer is empty, appropriate messages are displayed to prevent invalid operations.
* **Input Validation**: The program uses fgets () to safely read user input for lines of text, ensuring that the input does not exceed the buffer limit. It also handles incorrect menu choices by prompting the user to enter a valid choice.

**6. Areas for Improvement**

While the current implementation provides basic text editing functionality, several improvements could be made to enhance the user experience and functionality:

1. **Memory Management**:

* The program uses a fixed-size array (MAX\_LINES and MAX\_LENGTH) for storing lines of text. A more flexible approach could involve dynamic memory allocation using malloc() and realloc() to accommodate a variable number of lines with varying lengths.

1. **Undo/Redo Functionality**:

* Implementing undo and redo features would enhance the text editor's capabilities. Users could reverse or reapply their actions, making the program more powerful and user-friendly.

1. **Text Editing**:

* The current program only allows adding or deleting the last line. Adding the ability to edit specific lines or insert lines at particular positions would increase the flexibility of the editor.

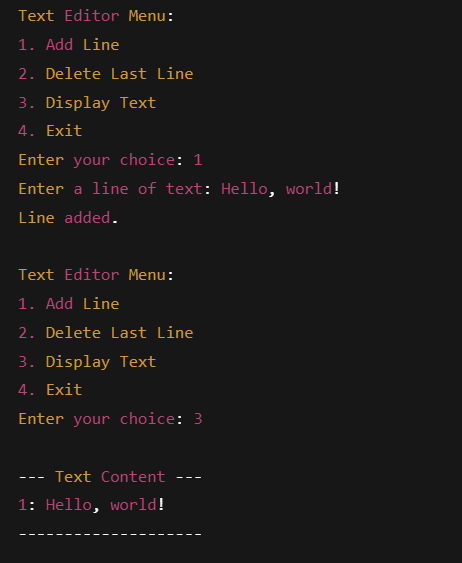
1. **File Operations**:

* The program currently does not save or load text from files. Implementing save (.txt file) and load functionality would allow users to persist their work.

1. **Error Handling for User Input**:

* The program could be enhanced by checking whether the user input for the menu choice is valid, ensuring that only numbers corresponding to available choices are accepted.

**Example of Program Output**

****

**7. Conclusion**

This simple text editor demonstrates basic principles of file handling, memory management, and user input in C. The program is efficient for handling small amounts of text and offers users the ability to add, delete, and display lines interactively. However, as outlined in the report, there are several opportunities to enhance the editor’s functionality, making it more robust and user-friendly. Future improvements such as dynamic memory allocation, advanced text editing, file I/O capabilities, and undo/redo features could transform this into a more complete text editing application.