**FATIMA JINNAH WOMEN UNIVERSITY**

****

**Department of Computer Science**

**Data Structure and Algorithm (LAB)**

**Submitted By:**

* Areeba Yasir(2023-BCS-016)
* Javeria Rahim(2023-BCS-046)
* Romaisa Zahid(2023-BCS-073)
* Sameen Noor(2023-BCS-077)

**Submitted To:**

Sir Adeel

**Project**

**Stack-Based Text Editor**

**Background**

Text editing is an essential part of digital interaction, whether it's coding, writing, or composing emails. As users work, they often need to make quick corrections or adjustments. Without the ability to undo mistakes, a simple error could lead to frustration and lost work. To address this, most modern text editors provide features like undo and redo, allowing users to move back and forth through their edits seamlessly.A stack-based text editor leverages the stack data structure to handle this history of changes. With each new operation (insert, delete, replace), the action is stored in a stack, enabling the user to undo the operation by simply "popping" it off the stack. This functionality adds flexibility and ease to the editing process, allowing users to experiment without the fear of making irreversible mistakes.The need for such a system becomes especially evident when working on long documents or complex code, where changes are frequent, and a simple way to reverse or repeat actions is invaluable.

### **Problem Statement:**

The challenge is to develop a text editor that supports dynamic text manipulation operations (insert, delete, replace) and allows users to undo and redo changes efficiently. The editor should also enable saving the document to a file, clearing the content, and displaying all lines.

In traditional text editors, managing undo and redo operations can be complex and memory-intensive, especially with array or list-based approaches. This project aims to provide an efficient, stack-based solution for tracking text changes, enabling users to:

* Undo the most recent change.
* Redo an undone change.
* Easily manipulate text lines (insert, delete, replace) using a linked list.
* Save content to an external text file.

### **Introduction:**

This stack-based text editor is a console application that uses stacks to manage operations and a linked list to store text. It supports inserting, deleting, and replacing lines, along with undo and redo functionality. The linked list ensures efficient text manipulation, while the stack tracks the sequence of operations.

### **Key Features:**

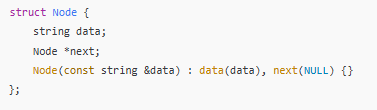
* **Insert Text:** Add text at a specific line.
* **Delete Line:** Remove a specific line.
* **Replace Text:** Modify an existing line's content.
* **Undo:** Revert the last action.
* **Redo:** Reapply the last undone action.
* **Save to File:** Save the document to a text file.
* **Clear Editor:** Remove all text from the editor.

### **Methodology:**

The text editor operates using two primary data structures:

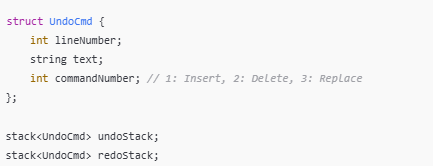
#### **Linked List:**

* Text is stored as nodes in a singly linked list, with each node holding one line of text.
* Efficient insertions and deletions are possible, especially at the head, tail, or a specific position.

****

#### **Stacks for Undo and Redo:**

* **Undo Stack:** Stores actions (insert, delete, replace) as command objects with operation type, line number, and text.
* **Redo Stack:** Stores actions that have been undone and allows reapplying them.



### **Steps of Operation:**

* **Inserting text:** The user specifies a line number and the text to insert. The text is then inserted at the head, tail, or a specific position in the linked list. After the insertion, the operation is pushed onto the undo stack.
* **Deleting text**: The user specifies a line number to delete. The specified line is removed from the list, and the operation is pushed onto the undo stack.
* **Replacing text:** The user specifies a line number and the new text. The old text is replaced with the new one, and the operation is pushed onto the undo stack.
* **Undo:** The most recent operation is popped from the undo stack, reversed, and the changes are pushed onto the redo stack.
* **Redo:** The most recent undone operation is popped from the redo stack and reapplied to the text document.
* **Saving to file:** The content of the editor is written to an output file (output.txt) using an ofstream.
* **Clearing the editor:** All lines in the linked list are deleted, and the editor is reset to an empty state.

### **Conclusion**

The stack-based text editor efficiently manages text editing operations with undo and redo functionality, enhancing user experience and control over document changes.