**Lab Sheet 5**

**Title:** Building a Simple Text Editor

**Introduction**

This project-based assignment integrates multiple data structures, including arrays, stacks, and queues, into the development of a simple text editor. This text editor will support basic text operations such as inserting text, deleting text, undo/redo functionality, and managing a clipboard for copy-paste operations. Through this project, students will learn to apply data structures in a real-world application scenario.

**Objective**

The objective of this assignment is to develop a simple text editor that uses arrays, stacks, and queues to manage its operations. Students will implement and apply these data structures to handle text editing, undo/redo functionality, and clipboard management.

**Problem Description**

1. **Text Editor Operations:**
   * Implement the text editor using an array to store the characters.
   * Implement a stack to manage the undo and redo operations.
   * Implement a queue for clipboard management to support copy-paste operations.
2. **Functional Requirements:**
   * Insert Text: Insert characters at a specified position.
   * Delete Text: Delete characters from a specified position.
   * Undo/Redo: Implement undo and redo functionalities using stacks.
   * Copy-Paste: Implement clipboard management using a queue to support copy and paste operations.

**Instructions**

1. **Text Editor Implementation:**
   * Create a class TextEditor that uses an array to store text characters.
   * Implement methods for inserting and deleting text.
   * Use stacks to manage undo and redo operations.
   * Use a queue to implement clipboard functionality.
2. **Insert Text:**
   * Method insertText(position, text) to insert text at the specified position.
3. **Delete Text:**
   * Method deleteText(position, length) to delete a specified number of characters from the specified position.
4. **Undo/Redo Operations:**
   * **Methods undo() and redo() to reverse and reapply the last editing operations.**
5. **Clipboard Management:**
   * Methods copy(position, length) and paste(position) to copy text to the clipboard and paste it at a specified position.

**Code:**

#include <iostream>

#include <stack>

#include <queue>

#include <string>

using namespace std;

class TextEditor {

private:

string text; // Using a string to store characters as an array

stack<string> undoStack; // Stack for undo operations

stack<string> redoStack; // Stack for redo operations

queue<string> clipboardQueue; // Queue for clipboard management

public:

// Insert text at a specified position

void insertText(int position, const string& newText) {

if (position < 0 || position > text.length()) {

cout << "Invalid position!" << endl;

return;

}

// Save current state for undo

undoStack.push(text);

while (!redoStack.empty()) redoStack.pop(); // Clear redo stack

text.insert(position, newText);

}

// Delete text starting from a specified position and with a specified length

void deleteText(int position, int length) {

if (position < 0 || position >= text.length() || length < 0) {

cout << "Invalid position or length!" << endl;

return;

}

// Save current state for undo

undoStack.push(text);

while (!redoStack.empty()) redoStack.pop(); // Clear redo stack

text.erase(position, length);

}

// Copy text to the clipboard (using queue)

void copy(int position, int length) {

if (position < 0 || position >= text.length() || length < 0 || position + length > text.length()) {

cout << "Invalid copy operation!" << endl;

return;

}

string copiedText = text.substr(position, length);

clipboardQueue.push(copiedText);

}

// Paste text from the clipboard at a specified position

void paste(int position) {

if (clipboardQueue.empty()) {

cout << "Clipboard is empty!" << endl;

return;

}

if (position < 0 || position > text.length()) {

cout << "Invalid position!" << endl;

return;

}

// Save current state for undo

undoStack.push(text);

while (!redoStack.empty()) redoStack.pop(); // Clear redo stack

string pastedText = clipboardQueue.front();

text.insert(position, pastedText);

}

// Undo the last operation

void undo() {

if (undoStack.empty()) {

cout << "Nothing to undo!" << endl;

return;

}

// Save current state for redo

redoStack.push(text);

// Revert to the previous state

text = undoStack.top();

undoStack.pop();

}

// Redo the last undone operation

void redo() {

if (redoStack.empty()) {

cout << "Nothing to redo!" << endl;

return;

}

// Save current state for undo

undoStack.push(text);

// Reapply the last undone state

text = redoStack.top();

redoStack.pop();

}

// Display the current text

void displayText() const {

cout << "Current Text: " << text << endl;

}

};

// Main function to demonstrate functionality

int main() {

TextEditor editor;

editor.insertText(0, "Hello World");

editor.displayText(); // Output: "Hello World"

editor.insertText(5, " Beautiful");

editor.displayText(); // Output: "Hello Beautiful World"

editor.deleteText(6, 10);

editor.displayText(); // Output: "Hello World"

editor.copy(0, 5); // Copy "Hello"

editor.paste(11);

editor.displayText(); // Output: "Hello WorldHello"

editor.undo();

editor.displayText(); // Output: "Hello World"

editor.redo();

editor.displayText(); // Output: "Hello WorldHello"

return 0;

}

**Report: Development of a Simple Text Editor Using Data Structures**

**1. Introduction**

In modern software applications, text editors play a crucial role by allowing users to create, modify, and manage text files efficiently. This project involves building a simple text editor that integrates various data structures, such as arrays, stacks, and queues, to perform essential text editing operations. The project serves as a practical exercise to demonstrate the application of fundamental data structures in real-world scenarios, enhancing students' understanding of how these structures can be utilized to solve complex problems efficiently.

**2. Objectives**

The primary objectives of this project are:

1. To develop a text editor capable of performing basic text editing operations, such as inserting and deleting text.
2. To implement undo/redo functionality using stacks.
3. To manage a clipboard for copy-paste operations using a queue.
4. To apply data structures like arrays, stacks, and queues to enhance the efficiency and functionality of the text editor.

**3. Problem Description**

The goal of this project is to design a simple text editor that supports the following operations:

1. **Insert Text**: Add characters at a specified position in the text.
2. **Delete Text**: Remove characters from a specified position.
3. **Undo/Redo**: Allow users to revert or reapply previous operations.
4. **Clipboard Management**: Enable copying and pasting text using a clipboard.

Each of these operations is implemented using appropriate data structures to ensure efficient performance.

**4. Data Structures Used**

**4.1 Arrays**

The text in the editor is stored using a dynamic array (represented as a string in C++). Arrays are chosen for their fast access times, allowing quick modifications to the text content. This makes inserting and deleting text efficient when performed at specific positions.

**4.2 Stacks**

Two stacks are used to implement the undo and redo functionalities:

* **Undo Stack**: Stores the state of the text before each modification. This allows the editor to revert changes by popping the previous state from the stack.
* **Redo Stack**: When an undo operation is performed, the current state is stored in the redo stack. This enables users to reapply changes if they choose to redo the action.

**4.3 Queue**

A queue is used to manage the clipboard for copy-paste operations. By storing copied text in a queue, the text editor can manage multiple copy-paste sequences in the order they were performed, ensuring efficient clipboard operations.

**5. Functional Requirements**

The text editor supports the following operations:

**5.1 Insert Text**

* Allows users to insert a string of text at any specified position.
* This operation uses the array to quickly modify the text.

**5.2 Delete Text**

* Enables users to delete a specified number of characters starting at a given position.
* Deleting text updates the array and adjusts the content accordingly.

**5.3 Undo/Redo Functionality**

* **Undo Operation**: Reverts the most recent text editing operation by restoring the text from the undo stack.
* **Redo Operation**: Reapplies the last undone operation by retrieving the text from the redo stack.
* Both operations are efficient due to the use of stacks, which follow a Last-In-First-Out (LIFO) order.

**5.4 Copy-Paste Operations**

* **Copy**: Extracts a portion of text from the specified position and stores it in the clipboard queue.
* **Paste**: Inserts the copied text at the desired position in the editor.
* The use of a queue for clipboard management ensures that multiple copy-paste operations are handled in sequence.

**6. Implementation Strategy**

To develop the text editor, the following implementation strategy was adopted:

1. **Designing Classes and Methods**: A class TextEditor was created to encapsulate all functionalities, including text management, undo/redo, and clipboard operations.
2. **Applying Data Structures**: Arrays were used for text storage, stacks for undo/redo, and a queue for clipboard management.
3. **Testing and Validation**: The text editor was tested with various input scenarios to ensure all functionalities worked as expected.

**7. Challenges Faced**

Some of the challenges encountered during the development process included:

* Ensuring efficient performance of the text editor, especially when dealing with large volumes of text.
* Implementing the undo and redo functionalities required careful management of the stack data structures to maintain the correct sequence of operations.
* Handling edge cases, such as inserting or deleting text at invalid positions, required additional validation.

**8. Results and Observations**

The text editor was successfully developed with the following capabilities:

* Users can efficiently insert and delete text at any position.
* The undo/redo functionality works seamlessly, allowing users to revert and reapply changes as needed.
* Copy-paste operations are managed effectively using a queue, supporting multiple clipboard entries.

The use of arrays, stacks, and queues significantly improved the performance and functionality of the editor, demonstrating the practical utility of these data structures in real-world applications.

**9. Conclusion**

The project achieved its goal of developing a simple yet functional text editor using C++. By leveraging arrays, stacks, and queues, the editor efficiently handles various text operations, highlighting the importance of data structures in software development. This project provided valuable insights into applying theoretical concepts to practical problems, enhancing the understanding of how data structures can be used to optimize performance.