**Lab Sheet 5**

Lab Assignment 5: Project-Based Application of Data Structures

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.

Test Cases and Expected Outputs

| Test Case | Input | Expected Output | Desired Output |
| --- | --- | --- | --- |
| Insert Text | insertText(0, "Hello") | "Hello" | "Hello" |
| Delete Text | insertText(0, "Hello"), deleteText(0, 2) | "llo" | "llo" |
| Undo Operation | insertText(0, "Hello"), deleteText(0, 2), undo() | "Hello" | "Hello" |
| Redo Operation | insertText(0, "Hello"), deleteText(0, 2), undo(), redo() | "llo" | "llo" |
| Copy-Paste | insertText(0, "Hello"), copy(0, 2), paste(5) | "HelloHe" | "HelloHe" |

Expected Outcomes

* Ability to apply arrays, stacks, and queues in a real-world application.
* Proficiency in implementing basic text editor operations.
* Understanding of managing undo/redo functionality using stacks.

**CLASS HEADER**

#include <iostream>

#include <vector>

#include <stack>

#include <queue>

#include <string>

class TextEditor {

private:

std::vector<char> text; // Array to store text characters

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

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

std::queue<std::string> clipboard; // Queue for clipboard management

public:

void insertText(int position, const std::string& newText);

void deleteText(int position, int length);

void undo();

void redo();

void copy(int position, int length);

void paste(int position);

void printText() const;

};

**INSERT**

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

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

std::cout << "Invalid position for insertion!" << std::endl;

return;

}

// Insert new text at the specified position

for (int i = 0; i < newText.size(); ++i) {

text.insert(text.begin() + position + i, newText[i]);

}

// Push the operation to the undo stack

undoStack.push("delete " + std::to\_string(position) + " " + std::to\_string(newText.size()));

// Clear the redo stack after a new operation

while (!redoStack.empty()) redoStack.pop();

}

**DELETE**

void TextEditor::deleteText(int position, int length) {

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

std::cout << "Invalid position or length for deletion!" << std::endl;

return;

}

// Capture the deleted text

std::string deletedText;

for (int i = 0; i < length; ++i) {

deletedText += text[position + i];

}

// Delete the specified characters

text.erase(text.begin() + position, text.begin() + position + length);

// Push the operation to the undo stack

undoStack.push("insert " + std::to\_string(position) + " " + deletedText);

// Clear the redo stack after a new operation

while (!redoStack.empty()) redoStack.pop();

}

**UNDO**

void TextEditor::undo() {

if (undoStack.empty()) {

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

return;

}

std::string operation = undoStack.top();

undoStack.pop();

redoStack.push(operation);

if (operation.find("insert") == 0) {

// Undo an insert operation

int position = std::stoi(operation.substr(7, operation.find(' ', 7) - 7));

int length = std::stoi(operation.substr(operation.find\_last\_of(' ') + 1));

text.erase(text.begin() + position, text.begin() + position + length);

} else if (operation.find("delete") == 0) {

// Undo a delete operation

int position = std::stoi(operation.substr(7, operation.find(' ', 7) - 7));

std::string data = operation.substr(operation.find\_last\_of(' ') + 1);

for (int i = 0; i < data.size(); ++i) {

text.insert(text.begin() + position + i, data[i]);

}

}

}

**REDO**

void TextEditor::redo() {

if (redoStack.empty()) {

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

return;

}

std::string operation = redoStack.top();

redoStack.pop();

undoStack.push(operation);

if (operation.find("insert") == 0) {

// Redo an insert operation

int position = std::stoi(operation.substr(7, operation.find(' ', 7) - 7));

std::string data = operation.substr(operation.find\_last\_of(' ') + 1);

for (int i = 0; i < data.size(); ++i) {

text.insert(text.begin() + position + i, data[i]);

}

} else if (operation.find("delete") == 0) {

// Redo a delete operation

int position = std::stoi(operation.substr(7, operation.find(' ', 7) - 7));

int length = std::stoi(operation.substr(operation.find\_last\_of(' ') + 1));

text.erase(text.begin() + position, text.begin() + position + length);

}

}

**COPY PASTE**

void TextEditor::copy(int position, int length) {

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

std::cout << "Invalid position or length for copy!" << std::endl;

return;

}

// Extract the text to copy

std::string copiedText;

for (int i = 0; i < length; ++i) {

copiedText += text[position + i];

}

// Add the copied text to the clipboard queue

clipboard.push(copiedText);

}

void TextEditor::paste(int position) {

if (clipboard.empty()) {

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

return;

}

std::string copiedText = clipboard.front();

clipboard.pop();

// Paste the text at the specified position

for (int i = 0; i < copiedText.size(); ++i) {

text.insert(text.begin() + position + i, copiedText[i]);

}

// Push the operation to the undo stack

undoStack.push("delete " + std::to\_string(position) + " " + std::to\_string(copiedText.size()));

}

**TEXT CASE CODE**

#include <iostream>

#include <vector>

#include <stack>

#include <queue>

#include <string>

class TextEditor {

private:

std::vector<char> text;

std::stack<std::string> undoStack;

std::stack<std::string> redoStack;

public:

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

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

std::cout << "Invalid position for insertion!" << std::endl;

return;

}

for (int i = 0; i < newText.size(); ++i) {

text.insert(text.begin() + position + i, newText[i]);

}

undoStack.push("delete " + std::to\_string(position) + " " + std::to\_string(newText.size()));

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

}

std::string getText() const {

return std::string(text.begin(), text.end());

}

};

int main() {

TextEditor editor;

editor.insertText(0, "Hello");

std::cout << "Current text: " << editor.getText() << std::endl; // Should output: Hello

return 0;

}

**EXPECTED OUTPUT**

Current text: Hello