**Assignment -2**

**Data Structure and Algorithms**

****

|  |  |
| --- | --- |
| Instructor Name | Dr Adnan Idrees |
| Students Names | Ahmed Yar |
| CMS ID | 480756 |
| Department | Computer Science |
| Semester | 3rd |

**Assignment No. 2**

1. **Introduction:**

The objective of this assignment is to deepen your understanding of stack data structures in C++ by implementing a simple word processing tool similar to Notepad. The tool will allow users to input, edit, and manage text, with a robust undo feature powered by a stack.

1. **Requirements:**

You will create a console-based application that enables users to perform basic text editing operations, with an emphasis on using a stack to manage undo functionality.

**Functional Specifications:**

1. **Text Input**: Allow users to input multiple lines of text.
2. **Edit Operations**:
   * **Add Text**: Users can add text to the current line or create a new line.
   * **Delete Text**: Users can delete text from the current line.
3. **Undo Feature**: Implement the ability to undo the last action (both add and delete) using a stack.
4. **Display Text**: Show the current text after each operation.
5. **Exit Option**: Allow users to exit the application gracefully.
6. Avoid using API calls, rather handle strings etc by developing your own methods.
7. If appropriate/needed, use Template classes as well.
8. Strictly fallow OPP for writing C++ code.
9. **Implementation Steps:**
10. **Define the Stack Class**:
    * Implement a stack class TextStack that holds text edit commands. The stack should support the following operations: (Hint: if appropriate use Template Class)
      + push(Command cmd): Adds a command to the stack.
      + pop(): Removes the most recent command from the stack.
      + top(): Retrieves the most recent command without removing it.
      + isEmpty(): Checks if the stack is empty.

Example structure for a command:

struct Command {

enum Operation { ADD, DELETE };

Operation op;

std::string text;

};

class Notepad {

private:

std::string text;

size\_t cursorPosition; // Current cursor position

CommandStack commandStack; // Stack to manage undo commands

Public:  
…..}

1. **Create the Notepad Class**:
   * Define a class Notepad that manages the current text and interacts with TextStack for undo operations.
   * Include the following attributes and methods:
     + A string to hold the current text.
     + An instance of TextStack to manage undo commands.
     + addText(const std::string &text): Adds text and pushes the command onto the stack.
     + deleteText(int length): Deletes the specified number of characters and pushes the command onto the stack.
     + undo(): Reverts the last action by utilizing the stack.
     + display(): Outputs the current text.
     + Avoid using API calls, rather handle strings etc by developing your own methods.
     + If appropriate/needed, use Template classes as well.
     + Strictly fallow OPP for writing C++ code.
2. **User Interface**:
   * Create a simple console menu that allows users to:
     + Develop a cursor showing the current position of character in a sentence.
     + Add text.
     + Delete text (specifying how many characters to delete).
     + Undo the last action.
     + Display the current text.
     + Exit the application.
     + Additional marks will be awarded to those who develop Windows Notepad like graphical interface with/without incorporating mouse handling.
3. **Error Handling**:
   * Implement error handling for invalid inputs (e.g., attempting to delete more characters than available, or trying to undo with an empty stack).

**Sample Menu:**

Welcome to Notepad-like Word Processor!

1. Add text

2. Delete text

3. Undo last action

4. Display current text

5. Exit

Please select an option (1-5):

**Deliverables:**

* A C++ source code file (notepad.cpp) implementing your tool.
* A brief report (1-2 pages) describing your design decisions, how the stack is utilized, and any challenges encountered during implementation.

**Submission Deadline:** November 8, 2024

Main.cpp

|  |
| --- |
| #include <conio.h>  // For getch()  #include <graphics.h>  #include <fstream>  #include <iostream>  #include <string>  #include "UI.cpp"  using namespace std;  // Main function  int main() {    int gd = DETECT, gm;    initgraph(&gd, &gm, nullptr);    string text;  // The current text    char choice;    while (true) {      displayMenu();      choice = getch();  // Get user input      switch (choice) {        case '1': {  // Create New Text          editText(text);          break;        }        case '2': {  // Save Text          saveText(text);          break;        }        case '3': {  // Exit Program          cleardevice();          setcolor(BLUE);          outtextxy(20, 100, const\_cast<char \*>("Exiting..."));          delay(2000);          closegraph();          return 0;        }        default: {  // Invalid input          cleardevice();          setcolor(RED);          outtextxy(20, 100, const\_cast<char \*>("Invalid choice. Try again."));          delay(1000);          break;        }      }    }    closegraph();    return 0;  } |

UI.cpp

|  |
| --- |
| #include <conio.h>  // For getch()  #include <graphics.h>  #include <fstream>  #include <iostream>  #include <string>  #include "stack.cpp"  #include "stack.h"  using namespace std;  struct Command {    enum Operation { ADD, REMOVE };    Operation op;    string text;  };  stack<Command> undoStack;  // Function to display the header with gradient effect  void displayHeader() {    setbkcolor(LIGHTCYAN);  // Light cyan background    setcolor(BLUE);         // Blue text color    cleardevice();    settextstyle(10, 0, 3);  // Triplex font, larger size for branding    outtextxy(10, 10, const\_cast<char \*>("Welcome Notepad!"));    setcolor(GREEN);                   // Green underline    line(10, 50, getmaxx() - 10, 50);  // Underline  }  // Function to display the main menu  void displayMenu() {    displayHeader();    settextstyle(8, 0, 2);  // Medium font for menu options    setcolor(BLACK);    outtextxy(40, 70, const\_cast<char \*>("Select an option below:"));    settextstyle(7, 0, 2);  // Different font for menu items    outtextxy(60, 120, const\_cast<char \*>("1. Create New Text"));    outtextxy(60, 160, const\_cast<char \*>("2. Save Current Text"));    outtextxy(60, 200, const\_cast<char \*>("3. Exit Program"));    settextstyle(8, 0, 1);  // Footer instruction    outtextxy(60, 260, const\_cast<char \*>("Use numbers (1-3) to select."));  }  // Function to display text with improved formatting  void displayText(const string &text) {    cleardevice();    displayHeader();    outtextxy(20, 60, const\_cast<char \*>("Current Text:"));    int y = 100;            // Start position for text display    settextstyle(3, 0, 1);  // Smaller font for text    setcolor(BLACK);    for (size\_t i = 0; i < text.length();         i += 40) {  // Display in chunks of 40 characters      string line = text.substr(i, 40);      outtextxy(20, y, const\_cast<char \*>(line.c\_str()));      y += 20;  // Move down for the next line    }    delay(3000);  // Pause for user to view the text  }  // Function to edit text  void editText(string &text) {    cleardevice();    displayHeader();    setcolor(DARKGRAY);    outtextxy(10, 70,              const\_cast<char \*>("Editing Mode: Type below (Enter to finish)"));    string buffer;    int pos = 0;    char ch;    while ((ch = getch()) != 13) {           // Enter key to finish editing      if (ch == 26 && !undoStack.empty()) {  // Ctrl+Z for undo        Command lastCommand = undoStack.top();        undoStack.pop();        if (lastCommand.op == Command::ADD) {          if (!buffer.empty()) {            buffer.pop\_back();            pos--;          }        } else if (lastCommand.op == Command::REMOVE) {          buffer += lastCommand.text;          pos++;        }      } else if (ch == 8 && pos > 0) {  // Backspace        pos--;        char deletedChar = buffer.back();        buffer.pop\_back();        Command removeCommand = {Command::REMOVE, string(1, deletedChar)};        undoStack.push(removeCommand);      } else if (pos < 1000 && ch != 8) {  // Add character        buffer += ch;        Command addCommand = {Command::ADD, string(1, ch)};        undoStack.push(addCommand);        pos++;      }      // Refresh the editor display      cleardevice();      displayHeader();      setcolor(BLACK);      outtextxy(10, 70,                const\_cast<char \*>("Editing Mode: Type below (Enter to finish)"));      outtextxy(20, 100, const\_cast<char \*>(buffer.c\_str()));    }    text = buffer;  }  // Function to save text to a file  void saveText(const string &text) {    ofstream file("notes.txt");    cleardevice();    displayHeader();    if (file.is\_open()) {      file << text;      file.close();      setcolor(DARKGRAY);      outtextxy(10, 60,                const\_cast<char \*>("Text successfully saved to 'notes.txt'."));    } else {      setcolor(RED);      outtextxy(10, 60, const\_cast<char \*>("Error: Could not save text."));    }    delay(2000);  // Pause for user to read the message  } |

Node.h

|  |
| --- |
| #ifndef NODE\_H  #define NODE\_H  template <class T>  class Node {    T data;    Node \*next;  // Pointer to the next node   public:    Node(T data) : data(data), next(nullptr) {}    // Default constructor    Node() : next(nullptr) {}    T getData() { return data; }    void setData(T data) { this->data = data; }    Node \*getNext() { return next; }    void setNext(Node \*next) { this->next = next; }  };  #endif |

Stack.h

|  |
| --- |
| #ifndef STACK\_H  #define STACK\_H  #include "Node.h"  template <class T>  class stack {    int size;    Node<T> \*Head;   public:    stack(int size);    stack();  // Default constructor    void push(T data);    void pop();    T top();    bool empty();  };  #endif |

Stack.cpp

|  |
| --- |
| #include "stack.h"  template <class T>  stack<T>::stack(int size) {    this->size = size;    Head = nullptr;  }  template <class T>  stack<T>::stack() {    Head = nullptr;  // Default constructor initializes Head to nullptr  }  template <class T>  void stack<T>::push(T data) {    Node<T> \*new\_node = new Node<T>(data);    new\_node->setNext(Head);    Head = new\_node;  }  template <class T>  void stack<T>::pop() {    if (Head != nullptr) {      Node<T> \*temp = Head;      Head = temp->getNext();      temp->setNext(nullptr);      delete temp;    } else {      throw std::runtime\_error("Stack is empty. Cannot pop.");    }  }  template <class T>  T stack<T>::top() {    if (Head != nullptr) {      return Head->getData();    } else {      throw std::runtime\_error("Stack is empty. Cannot get top.");    }  }  template <class T>  bool stack<T>::empty() {    return Head == nullptr;  } |