National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

School of Electrical Engineering and Computer Science

CS250 – Data Structures and Algorithms

Assignment 2: Stacks and Queues

Submission (Group) Details

Names	CMS ID
Muhammad Umer	345834
Muhammad Ahmed Mohsin	333060
Group	GP – 1
Instructor	Bostan Khan
Class	BEE12
Date	/03/2024

1 Table of Contents

2 Linked Lists

2.1 Objective

The objective of this assignment is to explore the applications of stacks and queues and gain hands-on experience with implementing and manipulating these data structures.

Please read the instructions for each task with great care and be sure to implement the requirements stated against each task.

This is a group assignment where each group can have a maximum of 2 students. Remember to include the names and registration numbers of both students at the start of the report.

Remember to document your work well in the documentation report. Explain the code, its design and test results with details for obtaining max marks for the whole assignment.

2.2 Deliverables

You will have to submit a separate .cpp source file for each task. Present the documentation report including the test results in a well-prepared PDF.

The report PDF and the sources files should be compressed in a single zip file and submitted on LMS with the name in following format: <Student_names>_<Reg_no>.zip

2.3 Documentation Report PDF

Provide a report PDF for all your linked list implementations, including explanations of the data structures and algorithms used, as well as any design decisions made for each task. Document the usage of your linked list in each of the application scenarios, including how the linked list data structure is utilized to solve the problem. **The marks of your coding tasks will be directly influenced by the documentation report.** Include clear instructions on how to compile/run your code and how to interact with the implemented applications.

2.4 Additional Notes:

- You are encouraged to be creative and innovative in your implementations. Consider additional features or optimizations that could enhance the functionality or performance of your applications.
- Collaboration with classmates is allowed for discussing concepts and problem-solving strategies, but each student must submit their own individual solution.
- **Plagiarism or copying of code from other students will result in zero marks.**

3 Tasks

3.1 Task 1

Implement a stack data structure in C++ to create a simple text editor with undo and redo functionality. The stack will be used to store the history of text changes, allowing users to undo their actions and redo them if needed.

Requirements:

1. Implement a stack data structure in C++ to store text strings.
2. Develop a text editor program that allows users to perform the following operations:
   a) *Insert text: Add a new text string to the editor.*
   b) *Delete text: Remove text from the editor.*
   c) *Undo: Revert the last text operation (insertion or deletion).*
   d) *Redo: Restore the text operation that was undone.*
3. Ensure that undo and redo operations are properly managed using the stack data structure.
4. Handle edge cases such as undo/redo when there are no actions to perform.
5. Implement a user-friendly menu-driven interface to interact with the text editor.

Documentation & Explanation

The `TextEditor` class uses a `Stack` to store the history of text changes, allowing users to undo their actions and redo them if needed. The `Stack` is implemented using a singly linked list to allow for dynamic memory allocation and deallocation. This is important because the number of text changes can vary, and the `Stack` should be able to handle this variation.

insert_text(string new_text)	Allows users to insert new text into the editor. The new text is added to the Stack using the push operation, and the previous text is stored for undo operations.
delete_text(int num_chars)	Allows users to delete text from the editor. The deleted text is added to the Stack using the push operation, and the previous text is stored for undo operations.
undo()	Reverts the last text operation (insertion or deletion) by popping the previous text from the Stack and pushing the current text to the redo Stack.
redo()	Restores the text operation that was undone by popping the previous text from the redo Stack and pushing the current text to the undo Stack.

Implementation

```cpp
#include <iostream>
#include <string>

using namespace std;

struct Node {
    string data;
    Node* next;
};

class Stack {
   private:
    Node* top;

   public:
    Stack() { top = nullptr; }

    void push(string data) {
        Node* new_node = new Node;
        new_node->data = data;
        new_node->next = top;
        top = new_node;
    }

    string pop() {
        if (top == nullptr) {
            return "Stack is empty!";
        }

        string data = top->data;
        Node* temp = top;
        top = top->next;
        delete temp;
        return data;
    }

    string peek() {
        if (top == nullptr) {
            return "Stack is empty!";
        }
        return top->data;
    }

    bool is_empty() { return top == nullptr; }
};

class TextEditor {
   private:
    Stack undo_stack;
    Stack redo_stack;
    string text = "";  // initial text

   public:
    void insert_text(string new_text) {
        undo_stack.push(text);
        text += new_text;  // two strings are concatenated
    }
```

```
    void delete_text(int num_chars) {
        if (num_chars > text.length()) {
            cout << "Invalid number of characters to delete!" << endl;
            return;
        }

        undo_stack.push(text);
        text = text.substr(
            0, text.length() - num_chars);  // remove last n characters
    }

    void undo() {
        if (undo_stack.is_empty()) {
            cout << "Nothing to undo!" << endl;
            return;
        }

        // push current text to redo stack and pop from undo stack
        redo_stack.push(text);
        text = undo_stack.pop();
        cout << "Undo successful!" << endl;
    }

    void redo() {
        if (redo_stack.is_empty()) {
            cout << "Nothing to redo!" << endl;
            return;
        }

        // push current text to undo stack and pop from redo stack
        undo_stack.push(text);
        text = redo_stack.pop();
        cout << "Redo successful!" << endl;
    }

    void display() { cout << text << endl; }
};
```

Testing

The following `main` function provides a user-friendly menu-driven interface to interact with the text editor, allowing users to perform insert, delete, undo, redo, and display operations.

```
void display_menu() {
    cout << "**== Text Editor Menu ==**" << endl;
    cout << "1) -> Insert text" << endl;
    cout << "2) -> Delete text" << endl;
    cout << "3) -> Undo" << endl;
    cout << "4) -> Redo" << endl;
    cout << "5) -> Display text" << endl;
    cout << "6) -> Exit" << endl;
}

int main() {
    TextEditor editor;
    int choice;
    string new_text;
    int num_chars;
    display_menu();
```

```
    while (true) {
        cout << "\nEnter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter text to insert: ";
                cin.ignore();  // clear input buffer
                getline(cin, new_text);
                editor.insert_text(new_text);
                break;
            case 2:
                cout << "Enter number of characters to delete: ";
                cin >> num_chars;
                editor.delete_text(num_chars);
                break;
            case 3:
                editor.undo();
                break;
            case 4:
                editor.redo();
                break;
            case 5:
                editor.display();
                break;
            case 6:
                cout << "Exiting the program ..." << endl;
                return 0;
            default:
                cout << "Invalid choice!" << endl;
        }
    }

    return 0;
}
```

Output

```
**== Text Editor Menu ==**
1) -> Insert text
2) -> Delete text
3) -> Undo
4) -> Redo
5) -> Display text
6) -> Exit

Enter your choice: 1
Enter text to insert: Initial text.

Enter your choice: 5
Initial text.

Enter your choice: 1
Enter text to insert:  More text.

Enter your choice: 5
Initial text. More text.
```

```
Enter your choice: 2
Enter number of characters to delete: 3

Enter your choice: 5
Initial text. More te

Enter your choice: 3
Undo successful!

Enter your choice: 5
Initial text. More text.

Enter your choice: 4
Redo successful!

Enter your choice: 5
Initial text. More te

Enter your choice: 6
Exiting the program ...
```

Instructions to Reproduce Output

Linux (Ubuntu 22.04)

```
- Compiler Version
gcc version 11.4.0 (Ubuntu 11.4.0-1ubuntu1~22.04)

- Steps
cd to extracted .zip directory
user@hostname:/dsa/assignment_2# g++ -o task_1 task_1.cpp
user@hostname:/dsa/assignment_2# ./task_1
```

Windows

```
- Compiler Version
gcc version 13.2.0 (Rev3, Built by MSYS2 project)

- Steps
cd to extracted .zip directory
C:\user\dsa\assignment_1> g++ -Wall -g task_1.cpp -o task_1.exe
C:\user\dsa\assignment_1> task_1.exe
```

3.2 Task 2

You are tasked with designing a printing queue management system for a busy office environment. The office has a single printer that serves multiple users. Users submit print jobs to the printer, and the printer processes the jobs in the order they were received. Each print job has a specific number of pages to be printed.

Requirements:

1. Implement a queue data structure in C++.
2. Implement functions to enqueue (add to the end of the queue) and dequeue (remove from the front of the queue).
3. Implement a function to display the current state of the queue.
4. Users should be able to submit print jobs to the queue with the number of pages to be printed.
5. The program should display the status of the printing queue, including the number of jobs in the queue and their details (e.g., job number, number of pages).
6. The printer should process print jobs in the order they were received, printing each job's pages and removing it from the queue once completed.
7. Display appropriate messages to inform users about their actions and the status of their print jobs.

Instructions:

a) Implement the Queue data structure using either arrays or linked lists while justifying your choice.
b) Design a menu with options for users like submit print jobs or exit the program.
c) Use appropriate error handling to prevent queue underflow or overflow.
d) Test your program with multiple scenarios to ensure its correctness.

Documentation & Explanation

The `Printer` class uses a `Queue` to store print jobs. The `Queue` is implemented using a linked list to allow for dynamic memory allocation and deallocation. This is important because the number of print jobs can vary, and the `Queue` should be able to handle this variation.

`submit_job``(string job, int pages)`	Allows users to submit print jobs to the queue with the number of pages to be printed. The job name and number of pages are added to the queue using the enqueue operation.
`process_job``()`	Processes print jobs in the order they were received, printing each job's pages and removing it from the queue once completed. The dequeue operation essentially removes the front print job from the queue, and the job is then printed.
`display_queue``()`	Displays the status of the printing queue, including the number of jobs in the queue and their details (e.g., job number, number of pages).

Design Justification

We opt for the Linked List-based implementation of the Queue for the following three reasons:

1. **Dynamic memory allocation:** A linked list allows for dynamic memory allocation, which is essential for handling a variable number of print jobs.
2. **Efficient insertion and deletion:** A singly linked list provides efficient insertion and deletion operations, which are important for managing the print queue.
3. **No fixed size:** A linked list does not have a fixed size, so it can grow or shrink based on the number of print jobs, unlike an array-based implementation.

Implementation

```
#include <iostream>
#include <string>

using namespace std;

struct Node {
    int pages;
    string job;
    Node *next;
};

class Queue {
   private:
    Node *front;
    Node *rear;

   public:
    Queue() {
        front = nullptr;
        rear = nullptr;
    }

    void enqueue(string job, int pages) {
        Node *new_node = new Node;
        new_node->job = job;
        new_node->pages = pages;
        new_node->next = nullptr;

        if (rear == nullptr) {
            front = new_node;
            rear = new_node;
        } else {
            rear->next = new_node;
            rear = new_node;
        }
    }

    string dequeue() {
        if (front == nullptr) {
            return "\0";
        }

        string job = front->job;
```

```
        Node *temp = front;
        front = front->next;
        delete temp;
        return job;
    }

    void display() {
        if (front == nullptr) {
            cout << "Queue is empty!" << endl;
            return;
        }

        Node *temp = front;
        int job_number = 1;
        cout << "Printing Queue:" << endl;
        cout << "----------------" << endl;
        while (temp != nullptr) {
            cout << "#" << job_number << " -> " << temp->job << " ("
                 << temp->pages << " pages)" << endl;
            temp = temp->next;
            job_number++;
        }
    }
};

class Printer {
   private:
    Queue print_queue;

   public:
    void submit_job(string job, int pages) {
        print_queue.enqueue(job, pages);
        cout << "Job " << job << " submitted!" << endl;
    }

    void process_job() {
        string job = print_queue.dequeue();
        if (job == "\0") {
            cout << "No jobs in the queue!" << endl;
            return;
        }
        cout << "Printing job (" << job << ") ..." << endl;
    }

    void display_queue() { print_queue.display(); }
};
```

Testing

The following `main` function demonstrates the usage of the `Printer` class. A menu is provided for users to submit print jobs, process print jobs, display the print queue, and exit the program.

```
void display_menu() {
    cout << "**== Printing Queue ==**" << endl;
    cout << "1) -> Submit print job" << endl;
    cout << "2) -> Process print job" << endl;
    cout << "3) -> Display print queue" << endl;
    cout << "4) -> Exit" << endl;
}
```

```
int main() {
    Printer printer;
    int choice;
    string job;
    int pages;
    display_menu();

    while (true) {
        cout << "\nEnter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter job name: ";
                cin >> job;
                cout << "Enter number of pages: ";
                cin >> pages;
                printer.submit_job(job, pages);
                break;
            case 2:
                printer.process_job();
                break;
            case 3:
                printer.display_queue();
                break;
            case 4:
                cout << "Exiting the program ..." << endl;
                return 0;
            default:
                cout << "Invalid choice!" << endl;
        }
    }
}
```

Output

```
**== Printing Queue ===**
1) -> Submit print job
2) -> Process print job
3) -> Display print queue
4) -> Exit

Enter your choice: 1
Enter job name: doc_a.pdf
Enter number of pages: 3
Job doc_a.pdf submitted!

Enter your choice: 1
Enter job name: doc_b.pdf
Enter number of pages: 5
Job doc_b.pdf submitted!

Enter your choice: 3
Printing Queue:
----------------
#1 -> doc_a.pdf (3 pages)
#2 -> doc_b.pdf (5 pages)

Enter your choice: 2
```

```
Printing job (doc_a.pdf) ...

Enter your choice: 2
Printing job (doc_b.pdf) ...

Enter your choice: 2
No jobs in the queue!

Enter your choice: 4
Exiting the program...
```

Instructions to Reproduce Output

Linux (Ubuntu 22.04)

```
- Compiler Version
gcc version 11.4.0 (Ubuntu 11.4.0-1ubuntu1~22.04)

- Steps
cd to extracted .zip directory
user@hostname:/dsa/assignment_2# g++ -o task_2 task_2.cpp
user@hostname:/dsa/assignment_2# ./task_2
```

Windows

```
- Compiler Version
gcc version 13.2.0 (Rev3, Built by MSYS2 project)

- Steps
cd to extracted .zip directory
C:\user\dsa\assignment_1> g++ -Wall -g task_2.cpp -o task_2.exe
C:\user\dsa\assignment_1> task_2.exe
```