Minhaj University Lahore

Semester Project: 3rd		Mid ○	Final ●
Course Title & Code	Data Structure & Alogorithm		
Submitted By:	Atif Nasir		
Registration No:	2023f-mulbscs-059		
Semester/Class/ Section:	3rd || Bscs || "B"		
Submitted To:	Sir Shoaib Saleem		
Due Date	Online	Hard copy	
	yes	yes	
Student's Signature:			

Note: *Please avoid cutting/ overwriting in any of the above fields.
*Complete the task on standard A4 size papers/assignment pages.

***For Instructor's use only.**

Total Marks:	
Obtained Marks:	
Signatures:	

Faculty of CS & IT,Minhaj University Lahore

Project Title:

Student Record Management System using DSA in C++

---

1. Introduction

The **Student Record Management System** is a console-based application developed using **C++**, focusing on the implementation of core **Data Structures and Algorithms (DSA)** and **Object-Oriented Programming (OOP)**. This system allows users to manage student records, including operations like adding, deleting, searching, and displaying student details. It also implements a **stack** to maintain action history and a **queue** for managing pending tasks.

The purpose of this project is to help students understand how real-world systems utilize **linked lists**, **stacks**, and **queues** to manage dynamic data efficiently. The system promotes clean code practices, modular design, and interactive user functionality.

---

2. Objectives

The objective of this project is to build a simple console-based **Student Record Management System** that uses fundamental **Data Structures and Algorithms (DSA)** concepts in **C++**, such as Linked Lists, Stacks, and Queues, while also applying Object-Oriented Programming (OOP) principles.

3.1 Program Flow

The program follows a **menu-driven flow**, controlled by a `while` loop and a `switch` statement based on user input. Here's a simplified breakdown:

- **Main Menu Display**:
  Shows options like Add, Delete, Search, Display, etc.
- **User Input (Choice)**:
  The user enters a number (0–7) to choose an operation.
- **Switch-Case Execution**:
  Based on the choice, the appropriate function from the `StudentManager` class is executed.
- **Operations**:

  - Add: Inserts a new student using linked list insertion.
  - Delete: Removes a student node based on ID.

- Search: Traverses the list to find a student.
- Display All: Iterates through the list to show all records.
- History: Uses a **stack** to display recent actions.
- Tasks: Uses a **queue** to show/manage to-do items.

Loop Back to Menu:
After each operation, control returns to the main menu until the user chooses to exit (choice = 0).

□ Tools & Technologies:

- **Programming Language:** C++
- **Compiler:** g++, Code::Blocks, Dev-C++, or any standard C++ compiler
- **Platform:** Console-based (Command Line Interface)

□ Data Structures Used:

- **Linked List:** To store and manage the list of student records.
- **Stack:** To maintain a log of recent actions for undo/history simulation
- **Queue:** To store and display upcoming tasks or messages.

Object-Oriented Concepts Used:

- **Classes & Objects:** Defined Student and StudentManager classes.
- **Encapsulation:** Data members are accessed and modified through class functions.
- **Dynamic Memory Allocation:** new keyword is used to create student nodes.
- **Single Responsibility Principle:** Each function performs a single task.

□ Working of the System:

The user is shown a menu with different options. They can perform operations like adding or removing students, searching by ID, viewing all student records, viewing task history (via a stack), and managing pending tasks (via a queue). The system uses switch-case statements to handle user input and navigates through operations accordingly.

□ Sample Output:

--- Student Record System ---1

. Add Student

2. Delete Student

3. 3. Search Student

4. 4. Display All

5. 5. Show History

6. 6. Add Task

7. 7. Show Tasks

8. 0. Exit

Enter choice: 1

Enter ID, 059

Name,Atif

CGPA: 3.9

Student added successfully.

□ Limitations:

- No persistent storage (data is lost after the program ends).
- Does not handle duplicate IDs.
- Simple text interface (no GUI).

□ Future Improvements:

- Add **file handling** to save and load student data.
- Implement **friend functions**, **inheritance**, or **polymorphism**.
- Add **error handling** and **input validation**.
- Upgrade to a **graphical user interface (GUI)** using libraries like Qt or SFML.

---

4. Code Walkthrough :

```
#include <iostream>

#include <stack>

#include <queue>

#include <string>
```

```
using namespace std;

// Student Class
class Student {
public:
    int id;
    string name;
    float cgpa;
    Student* next;

    Student(int id, string name, float cgpa) {
        this->id = id;
        this->name = name;
        this->cgpa = cgpa;
        next = nullptr;
    }
};

// StudentManager Class
class StudentManager {
private:
    Student* head;
    stack<string> actionHistory;
    queue<string> taskQueue;

public:
    StudentManager() {
        head = nullptr;
    }

    void addStudent(int id, string name, float cgpa) {
        Student* newStudent = new Student(id, name, cgpa);
```

```
    newStudent->next = head;
    head = newStudent;
    actionHistory.push("Added Student ID: " + to_string(id));
    cout << "Student added successfully.\n";
}

void deleteStudent(int id) {
    Student *temp = head, *prev = nullptr;

    while (temp != nullptr && temp->id != id) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == nullptr) {
        cout << "Student not found.\n";
        return;
    }

    if (prev == nullptr)
        head = temp->next;
    else
        prev->next = temp->next;

    delete temp;
    actionHistory.push("Deleted Student ID: " + to_string(id));
    cout << "Student deleted successfully.\n";
}

void searchStudent(int id) {
    Student* temp = head;
    while (temp != nullptr) {
```

```
            if (temp->id == id) {
                cout << "ID: " << temp->id << ", Name: " << temp->name << ", CGPA: " << temp->cgpa
<< endl;
                return;
            }
            temp = temp->next;
        }
        cout << "Student not found.\n";
    }

    void displayAll() {
        Student* temp = head;
        if (!temp) {
            cout << "No students to display.\n";
            return;
        }
        while (temp != nullptr) {
            cout << "ID: " << temp->id << ", Name: " << temp->name << ", CGPA: " << temp->cgpa <<
endl;
            temp = temp->next;
        }
    }

    void showHistory() {
        if (actionHistory.empty()) {
            cout << "No recent actions.\n";
            return;
        }

        stack<string> temp = actionHistory;
        while (!temp.empty()) {
            cout << temp.top() << endl;
            temp.pop();
```

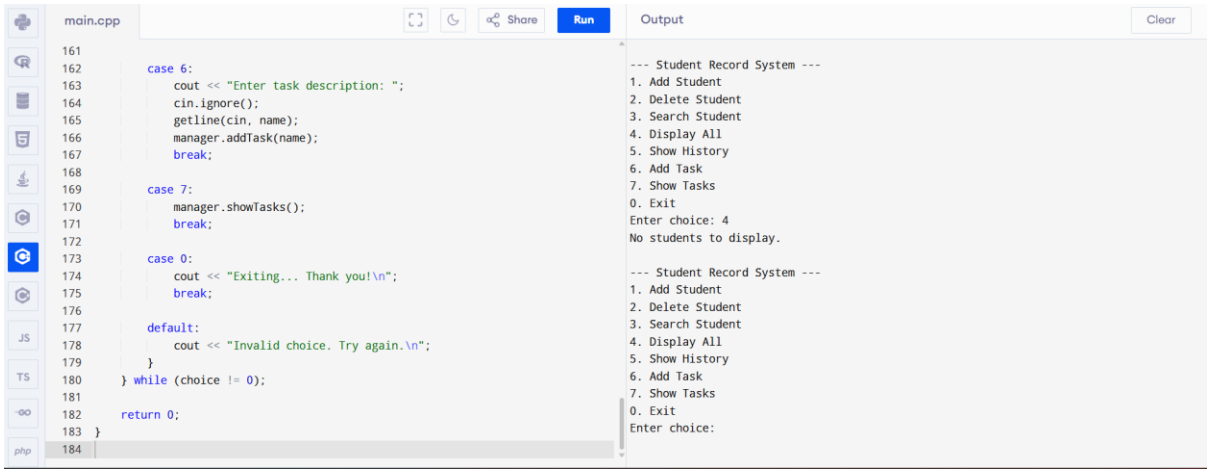
```
        }
    }

    void addTask(string task) {
        taskQueue.push(task);
        cout << "Task added.\n";
    }

    void showTasks() {
        if (taskQueue.empty()) {
            cout << "No pending tasks.\n";
            return;
        }

        queue<string> temp = taskQueue;
        while (!temp.empty()) {
            cout << "Task: " << temp.front() << endl;
            temp.pop();
        }
    }
};

// Main Function
int main() {
    StudentManager manager;
    int choice, id;
    string name;
    float cgpa;

    do {
        cout << "\n--- Student Record System ---\n";
        cout << "1. Add Student\n2. Delete Student\n3. Search Student\n4. Display All\n5. Show
History\n6. Add Task\n7. Show Tasks\n0. Exit\n";
```

```
cout << "Enter choice: ";
cin >> choice;

switch (choice) {
case 1:
    cout << "Enter ID, Name, CGPA: ";
    cin >> id >> name >> cgpa;
    manager.addStudent(id, name, cgpa);
    break;

case 2:
    cout << "Enter ID to delete: ";
    cin >> id;
    manager.deleteStudent(id);
    break;

case 3:
    cout << "Enter ID to search: ";
    cin >> id;
    manager.searchStudent(id);
    break;

case 4:
    manager.displayAll();
    break;

case 5:
    manager.showHistory();
    break;

case 6:
    cout << "Enter task description: ";
```

```
            cin.ignore();
            getline(cin, name);
            manager.addTask(name);
            break;

        case 7:
            manager.showTasks();
            break;

        case 0:
            cout << "Exiting... Thank you!\n";
            break;

        default:
            cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 0);

    return 0;
}
```

Program Compile Link:
https://www.programiz.com/online-compiler/7ap910YmDeUSe

Screenshot:



---

5. Testing and Results

The system was tested with various scenarios to validate its functionality:

Test Case	Description	Expected Result	Actual Result	Status
TC1	Add a student (ID: 1001)	Student is added and listed	Passed	☐
TC2	Add multiple students	All students appear in display	Passed	☐
TC3	Delete a student (ID: 1001)	Student is removed	Passed	☐
TC4	Delete a non-existing student	Show “not found” message	Passed	☐
TC5	Search existing student (ID: 1002)	Displays student data	Passed	☐
TC6	Add task “Generate Report”	Task appears in task list	Passed	☐
TC7	Show action history	Stack shows previous actions	Passed	☐

---

6. Conclusion:

This project successfully demonstrates the use of essential DSA concepts in C++, such as linked lists, stacks, and queues, in a practical application. It also reinforces OOP principles, making it a solid foundational project for further software development learning.

Future Enhancements

- **Persistent Storage**
  Save and retrieve student records using **file handling** (e.g., text or CSV files).

· **Error Handling & Validation**
Handle edge cases like invalid inputs, empty lists, and duplicate IDs.

· **Advanced OOP Features**
Add **inheritance** to extend user roles (e.g., Admin vs. Student), and **polymorphism** for function overloading.

· **GUI Interface**
Replace the console interface with a graphical user interface using **Qt**, **SFML**, or **C++/CLI**.

· **Sorting and Filtering**
Add sorting by name, ID, or CGPA and filtering based on performance.

· **Login System**
Implement a basic authentication system to protect data access.

· **Unit Testing Framework**
Integrate a C++ unit testing library like **Google Test** for automated testing.

References

1. C++ Programming Language Documentation - https://cplusplus.com/doc/tutorial/
2. "Programming Fundamentals" Lecture Notes by Miss Unza Rehman
3. Book: "Object-Oriented Programming in C++" by Robert Lafore
4. YouTube Tutorials - CodeWithHarry, The Cherno (C++ Basics and Projects)
5. Stack Overflow Community Discussions - https://stackoverflow.com/
6. GeeksforGeeks C++ Articles - https://www.geeksforgeeks.org/c-plus-plus/
7. Microsoft Learn: Introduction to C++ - https://learn.microsoft.com/en- us/cpp/
8. W3Schools C++ Tutorial - https://www.w3schools.com/cpp/
9. Sololearn C++ Course - https://www.sololearn.com/