SCHOOL MANAGEMENT SYSTEM

Project management to the

SRM University- AP ,Andhra Pradesh

For the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Abhiram (AP23110010260)
Tejaswi(AP23110010276)
Lakshmi swetha(AP23110010286)
Bala Manindra(AP23110010293)

Under the Guidance of
Mrs. Karnena Kavitha Rani

SRM University - AP

Neerukonda, Mangalagiri, Guntur Andhra Pradesh - 522 240

[November, 2024]

Certificate

Date: November-2024

This is to certify that the work present in this Project entitled "student management system" has been carried out by group-5 under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in School of Engineering and Sciences.

Supervisor

Prof. / Mrs. Karnena Kavitha Rani

Assistant Professor.

Department of Computer Science.

Acknowledgements

We would like to express our gratitude to **Mrs. Karnena Kavitha Rani** for assisting us with the research at every step. Without the advice and assistance of our adviser, this study effort would not be feasible. The suggestions, feedback, and encouragement Our advisors were extremely helpful in assisting us in finishing this research.

Table of contents **Page No:**

Abstract

The "Student Management System using Doubly Linked List" is a C++ project designed to manage and organize student-related information within an educational institution. This system leverages a doubly linked list data structure to store, retrieve, and manipulate student records efficiently. The application offers a user-friendly interface through the console, enabling administrators to perform key operations such as adding new student records, removing existing records, searching for students, and displaying student details. The system supports detailed viewing of student information, which can be filtered department-wise or course-wise, covering courses such as BSc, BTech, MSc, MTech, and PhD. The use of a doubly linked list allows for efficient traversal and modification of records in both forward and backward directions, ensuring flexible data management. This system provides a robust and structured approach for educational institutions to maintain comprehensive student data, improving overall record handling and operational efficiency.

1.Introduction

The "Student Management System using Doubly Linked List" is a C++ project for efficiently managing student records. It supports adding, removing, searching, and displaying students by department or course, using a doubly linked list for effective data handling.

1.1 Core Classes:

The core classes include Student, representing individual student details (name, ID, age, department, course), and Student Management System, which manages student records using a doubly linked list. Key methods include adding, removing, searching, displaying, and updating student data, enabling efficient bi-directional data handling.

1.2 Key Features:

The system includes adding and removing student records, searching for specific students, and displaying details based on department or course (e.g., BSc, BTech, MSc, MTech, PhD). The use of a doubly linked list allows bi-directional traversal, enabling efficient access and management of student data.

1.3 Security Measures:

It is implemented to protect data integrity and access. Basic authentication ensures only authorized users can modify or view records, while validation checks prevent incorrect or duplicate entries. Additionally, data encryption safeguards sensitive information, and error handling mechanisms maintain system stability and prevent data corruption.

2. Methodology

The methodology involves creating a Student class with student details and pointers for a doubly linked list. The StudentList class manages operations like adding, removing, searching, and displaying students. Students are added at the end, and removals adjust pointers. Display and search functions traverse the list, and a destructor ensures memory is freed. A user menu facilitates interaction for managing student data.

2.1 Modular Design:

1. Student Class (Node):

This class represents each student as a node in the doubly linked list.

Attributes:

Student ID

Name

Department

Course

Age, Contact Info, etc.

Methods:

Constructor to initialize the student details.

Setters and getters for each attribute.

2. DoublyLinkedList Class:

This class manages the doubly linked list of students.

Attributes:

head (points to the first student node)

tail (points to the last student node)

Methods:

addStudent(student) – Adds a new student to the list.

removeStudent(studentID) – Removes the student with the given ID from the list.

searchStudent(studentID) – Searches for a student by ID.

displayDepartmentWise(department) – Displays all students in a particular department.

displayCourseWise(course) – Displays all students in a particular course.

displayAll() – Displays all students in the list.

3. StudentManagementSystem Class:

This is the main driver class that interacts with the user and manages the student records.

Attributes:

Instance of DoublyLinkedList (to store all students).

Methods:

addNewStudent() – Collects student data from the user and adds a new student.

removeExistingStudent() – Prompts user for student ID and removes the student.

searchStudentDetails() – Prompts for student ID and displays details if found.

showStudentsByDepartment() – Prompts for department name and displays students.

showStudentsByCourse() – Prompts for course type and displays students.

4. Utility Functions:

Input validation: Functions for validating inputs such as student IDs, names, and courses.

Sorting/Filtering: Optional feature to sort students by name or ID and filter by department/course.

3. Discussion

3.1 CODE

```
#include <iostream>
#include <string>
using namespace std;
class Student {
public:
   string roll_number;
   string name;
   string department;
   string course;
   Student* prev;
   Student* next;
   Student(const string& roll_number, const string& name,
const string& department, const string& course)
      : roll_number(roll_number), name(name),
department(department), course(course), prev(nullptr),
next(nullptr) {}
};
class StudentList {
private:
   Student* head;
public:
   StudentList() : head(nullptr) {}
   void addStudent(const string& roll_number, const string&
name, const string& department, const string& course) {
      Student* newStudent = new Student(roll_number, name,
department, course);
      if (!head) {
```

```
        head = newStudent;
    } else {
        Student* current = head;
        while (current->next) {
            current = current->next;
        }
        current->next = newStudent;
        newStudent->prev = current;
    }
    cout << "Student added successfully!\n";
}
void removeStudent(const string& roll_number) {
    Student* current = head;
    while (current && current->roll_number != roll_number)
{
        current = current->next;
    }
    if (!current) {
        cout << "Student with roll number " << roll_number <<
" not found!\n";
        return;
    }
    if (current->prev) {
        current->prev->next = current->next;
    } else {
        head = current->next;
    }
    if (current->next) {
        current->next->prev = current->prev;
    }
    delete current;
    cout << "Student with roll number " << roll_number << "
removed successfully!\n";
}
```

```
  void displayStudents() const {
    if (!head) {
      cout << "No students to display.\n";
      return;
    }
    cout << "Student Details:\n";
    Student* current = head;
    while (current) {
      cout << "Roll Number: " << current->roll_number
        << ", Name: " << current->name
        << ", Department: " << current->department
        << ", Course: " << current->course << "\n";
      current = current->next;
    }
  }
  void searchStudent(const string& roll_number) const {
    Student* current = head;
    while (current) {
      if (current->roll_number == roll_number) {
        cout << "Student Found:\n";
        cout << "Roll Number: " << current->roll_number
          << ", Name: " << current->name
          << ", Department: " << current->department
          << ", Course: " << current->course << "\n";
        return;
      }
      current = current->next;
    }
    cout << "Student with roll number " << roll_number << "
not found!\n";
  }
  ~StudentList() {
    while (head) {
      Student* temp = head;
```

```
            head = head->next;
            delete temp;
        }
    }
};
int main() {
    StudentList list;
    int choice;
    string roll_number, name, department, course;
    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Add Student\n";
        cout << "2. Remove Student\n";
        cout << "3. Display All Students\n";
        cout << "4. Search Student\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter Roll Number: ";
                cin >> roll_number;
                cin.ignore(); // Ignore leftover newline character
                cout << "Enter Name: ";
                getline(cin, name);
                cout << "Enter Department: ";
                getline(cin, department);
                cout << "Enter Course: ";
                getline(cin, course);
                list.addStudent(roll_number, name, department,
course);
                break;
            case 2:
                cout << "Enter Roll Number to remove: ";
```

```
            cin >> roll_number;
            list.removeStudent(roll_number);
            break;
        case 3:
            list.displayStudents();
            break;
        case 4:
            cout << "Enter Roll Number to search: ";
            cin >> roll_number;
            list.searchStudent(roll_number);
            break;
        case 5:
            cout << "Exiting program...\n";
            return 0;

        default:
            cout << "Invalid choice! Please enter a valid option.\n";
    }
  }

  return 0;
}
```

3.2 Output of Code

```
Menu:
1. Add Student
2. Remove Student
3. Display All Students
4. Search Student
5. Exit
Enter your choice: 1
```

Enter Roll Number: AP23110010260
Enter Name: Abhiram
Enter Department: CSE
Enter Course: Btech
Student added successfully!

Menu:
1. Add Student
2. Remove Student
3. Display All Students
4. Search Student
5. Exit
Enter your choice: 1
Enter Roll Number: AP23110010293
Enter Name: Manindra
Enter Department: CSE
Enter Course: btech
Student added successfully!

Menu:
1. Add Student
2. Remove Student
3. Display All Students
4. Search Student
5. Exit
Enter your choice: 2
Enter Roll Number to remove:
AP23110010293
Student with roll number AP23110010293 removed successfully!

Menu:
1. Add Student
2. Remove Student

3. Display All Students
4. Search Student
5. Exit
Enter your choice: 4
Enter Roll Number to search: AP23110010260
Student Found:
Roll Number: AP23110010260, Name: Abhiram, Department: CSE, Course: Btech

Menu:
1. Add Student
2. Remove Student
3. Display All Students
4. Search Student
5. Exit
Enter your choice: 3
Student Details:
Roll Number: AP23110010260, Name: Abhiram, Department: CSE, Course: Btech

Code Explanation:

This C++ program implements a **doubly linked list** to manage a list of students. Each student has attributes such as roll number, name, department, and course. Here is a detailed explanation of the code:

Class Definitions

- **`Student` Class:**
  - Represents a single student with the following attributes:
  - `roll_number` (string): Unique identifier for a student.
  - `name` (string): Student's name.

    - `department` (string): Department name.
    - `course` (string): Course name.
      - Pointers `prev` and `next` allow the student object to link to the previous and next students in the list.
      - Constructor initializes the attributes.
- **`StudentList` Class:**
    - Manages the list of students using the `head` pointer, which points to the first student in the list.

Class Methods

1. `addStudent`:

- Adds a new student to the end of the doubly linked list.
- If the list is empty (`head == nullptr`), the new student becomes the `head`.
- Otherwise, traverses the list to the last student and links the new student to it.

2. `removeStudent`:

- Removes a student based on their roll number.
- Traverses the list to find the matching student.
- Updates the links (`prev` and `next`) of adjacent nodes to bypass the removed student.
- Deletes the student from memory and handles edge cases (e.g., removing the head).

3. `displayStudents`:

- Traverses the list and displays details of all students.
- Handles the case when the list is empty.

4. `searchStudent`:

- Searches for a student by their roll number.
- If found, displays the student details; otherwise, informs the user that the student is not found.

5. Destructor (`~StudentList`):

- Deallocates memory for all student nodes in the list, ensuring no memory leaks.

`Main` Function

1. Displays a **menu** to interact with the program:
    - **Add Student**: Prompts the user for student details and adds a new student.
    - **Remove Student**: Removes a student by roll number.
    - **Display All Students**: Shows details of all students in the list.
    - **Search Student**: Searches and displays details of a specific student by roll number.
    - **Exit**: Ends the program.
2. Handles user input and calls appropriate methods.

Notable Features

1. **Doubly Linked List**:
    - Allows efficient traversal and removal from both ends of the list.
    - Links (`prev` and `next`) are updated properly when adding or removing nodes.
2. **Dynamic Memory Management**:
    - The `new` keyword is used to allocate memory for students.
    - The destructor (`~StudentList`) ensures memory is freed when the program exits.

3. **String Handling**:
    - `getline()` is used for strings with spaces (e.g., name or department).
4. **Menu Loop**:
    - A `while (true)` loop keeps the menu active until the user chooses to exit.

Potential Enhancements

- **Input Validation**:
    - Check for duplicate roll numbers when adding students.
    - Validate user input for menu choices to avoid invalid input (non-integer values).
- **Sorting**:
    - Add functionality to display students in sorted order by roll number or name.
- **Improved Search**:
    - Enhance search functionality for partial matches (e.g., part of the name).

4. Concluding Remarks

The Student Management System using a doubly linked list efficiently handles operations such as adding, removing, searching, and displaying student data. This structure allows for bi-directional traversal, making data management straightforward and robust. The modular design enhances code clarity, maintainability, and scalability, ensuring that each function can be maintained or extended with minimal impact on other components.

5. Future Work

5.1 Enhanced User Authentication:

Integrate authentication to restrict access, ensuring only authorized personnel can manage student records.

5.2 Expanded Functionality:

Bulk Upload: Add functionality to upload student details in bulk using files (e.g., CSV or Excel).
Data Export: Allow data export for reports or data analysis.

5.3 Security Measures:

Implement data validation, encryption for sensitive data, and secure data storage to protect student information.

5.4 User Experience Enhancements:

Create a more user-friendly console interface or consider a GUI version.
Provide error handling and feedback messages for a smooth user experience.

5.5 Additional Features for Users:

Sorting Options: Sort and display students based on various attributes (e.g., alphabetical, department-wise).
Reporting: Generate printable reports of students by department or course.

This framework offers a comprehensive guide to developing a Student Management System using a doubly linked list in C++.

References

1. Thinking in C++, Bruce, Eckel, Pearson, Second edition, Volume 1, 2002.

2. Object-oriented programming in C++, Robert Lafore, Course Sams Publishing, Fourth edition,2001.

3. Lischner, Ray. STL Pocket Reference: Containers, Iterators, and Algorithms. " O'Reilly Media,Inc.", 2003.