Data Structures And Algorithms

Submitted By:

Jalal Saleem(2022-EE-275)
Saad Ijaz (2022-EE-294)
Muhammad Abdullah Idrees (2022-EE-304)
Muhammad Zohaib (2022-EE-299)

Submitted To:

Sir Dr. Haris Anwar

DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF ENGINEERING & TECHNOLOGY, LAHORE

Table of Contents

Introduction

The Student Learning Management System (LMS) is a console-based application for managing student records. It allows users to add, delete, search, update, display, sort, save, and load student data. The system uses a linked list and hash map for efficient data management and quick access to student records.

Classes and Their Functions

Student

The `Student` class represents a student and contains the following attributes:

- `name`: Student's name.
- `rollNumber`: Student's roll number.
- `department`: Student's department.
- `section`: Student's section.
- `cgpa`: Student's CGPA.
- `next`: Pointer to the next student in the linked list.

Constructor:

```
Student(string name, int rollNumber, string department, string section,
float cgpa)
```

StudentList

The `StudentList` class manages a linked list of students and provides various operations to manipulate the student records.

Attributes:

- `head`: Pointer to the head of the linked list.
- `rollNumberMap`: Unordered map for quick search by roll number.

Methods:

- `addStudent(string name, int rollNumber, string department, string section, float cgpa)`: Adds a new student to the list.
- `deleteStudent(int rollNumber)`: Deletes a student by roll number.
- `searchStudent(int rollNumber)`: Searches for a student by roll number.
- `updateStudent(int rollNumber, string name, string department, string section, float cgpa)`: Updates a student's details.
- `displayAllStudents()`: Displays all students.
- `sortStudentsByCGPA()`: Sorts students by CGPA.
- `sortStudentsByName()`: Sorts students by name.

StudentLMS

The `StudentLMS` class provides a user interface to interact with the `StudentList`.

Methods:

- `addStudent()`: Prompts the user to add a student.
- `deleteStudent()`: Prompts the user to delete a student.
- `searchStudent()`: Prompts the user to search for a student.
- `updateStudent()`: Prompts the user to update a student's details.
- `displayAllStudents()`: Displays all students.
- `sortStudentsByCGPA()`: Sorts students by CGPA.
- `sortStudentsByName()`: Sorts students by name.
- `saveToFile()`: Saves student records to a file.
- `loadFromFile()`: Loads student records from a file.

Features

Add Student

This feature allows the user to add a new student to the system. The user will be prompted to enter the student's name, roll number, department, section, and CGPA.

Delete Student

This feature allows the user to delete a student from the system by entering the roll number of the student to be deleted.

Search Student

This feature allows the user to search for a student by roll number. If the student is found, their details will be displayed.

Update Student

This feature allows the user to update the details of an existing student by entering the roll number and the new details.

Display All Students

This feature displays the details of all students in the system.

Sort Students by CGPA

This feature sorts the students in the system by their CGPA in ascending order.

Code:

```
#include <iostream> // Input/output stream
#include <fstream> // File stream
#include <string> // String operations
#include <unordered_map> // Unordered map container
#include <functional> // Function objects

using namespace std;

// Student class
class Student {
public:
    string name;
    int rollNumber;
    string department;
    string section;
    float cgpa;

    Student *next;

    // Constructor
    Student(string name, int rollNumber, string department, string section,
float cgpa)
        : name(name), rollNumber(rollNumber), department(department),
section(section), cgpa(cgpa), next(nullptr) {}
};

// StudentList class
class StudentList {
```

```
private:

public:
    StudentList() : head(nullptr) {}
    Student *head;
    unordered_map<int, Student *> rollNumberMap; // HashMap for quick search
by roll number

    // Add student
    void addStudent(string name, int rollNumber, string department, string
section, float cgpa) {
        Student *newStudent = new Student(name, rollNumber, department,
section, cgpa);
        newStudent->next = head;
        head = newStudent;
        rollNumberMap[rollNumber] = newStudent;
    }

    // Delete student by roll number
    void deleteStudent(int rollNumber) {
        Student *temp = head;
        Student *prev = nullptr;

        if (temp != nullptr && temp->rollNumber == rollNumber) {
            head = temp->next;
            rollNumberMap.erase(rollNumber);
            delete temp;
            return;
        }

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

        if (temp == nullptr)
            return;

        prev->next = temp->next;
        rollNumberMap.erase(rollNumber);
        delete temp;
    }

    // Search student by roll number
    Student *searchStudent(int rollNumber) {
        if (rollNumberMap.find(rollNumber) != rollNumberMap.end()) {
            return rollNumberMap[rollNumber];
        }
        return nullptr;
    }

    // Update student details
    void updateStudent(int rollNumber, string name, string department, string
section, float cgpa) {
        Student *student = searchStudent(rollNumber);
        if (student) {
            student->name = name;
            student->department = department;
            student->section = section;
            student->cgpa = cgpa;
        }
    }

    // Display all students
    void displayAllStudents() {
        Student *temp = head;
        while (temp != nullptr) {
            cout << "Name: " << temp->name << ", Roll Number: " << temp-
>rollNumber
                 << ", Department: " << temp->department << ", Section: " <<
temp->section
                 << ", CGPA: " << temp->cgpa << endl;
            temp = temp->next;
        }
    }

    // Sort students by CGPA
    void sortStudentsByCGPA() {
        if (!head)
            return;

        head = mergeSort(head, [](Student *a, Student *b) { return a->cgpa <
b->cgpa; });
    }

    // Sort students by name
    void sortStudentsByName() {
        if (!head)
            return;

        head = mergeSort(head, [](Student *a, Student *b) { return a->name <
b->name; });
    }

private:
    // Merge sort helper
    Student *mergeSort(Student *head, function<bool(Student *, Student *)>
compare) {
        if (!head || !head->next)
            return head;

        Student *middle = getMiddle(head);
        Student *nextToMiddle = middle->next;

        middle->next = nullptr;

        Student *left = mergeSort(head, compare);
        Student *right = mergeSort(nextToMiddle, compare);

        return merge(left, right, compare);
    }

    // Merge two sorted lists
    Student *merge(Student *left, Student *right, function<bool(Student *,
Student *)> compare) {
        if (!left)
            return right;
        if (!right)
            return left;

        Student *result = nullptr;
        if (compare(left, right)) {
            result = left;
            result->next = merge(left->next, right, compare);
        } else {
            result = right;
            result->next = merge(left, right->next, compare);
        }

        return result;
    }

    // Get middle of the linked list
    Student *getMiddle(Student *head) {
        if (!head)
            return head;

        Student *slow = head;
        Student *fast = head->next;

        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
};

// StudentLMS class
class StudentLMS {
private:
    StudentList studentList;

public:
    void addStudent() {
        string name, department, section;
        int rollNumber;
        float cgpa;
        cout << "Enter name: ";
        cin >> name;
        cout << "Enter roll number: ";
        cin >> rollNumber;
        cout << "Enter department: ";
        cin >> department;
        cout << "Enter section: ";
        cin >> section;
        cout << "Enter CGPA: ";
        cin >> cgpa;

        studentList.addStudent(name, rollNumber, department, section, cgpa);
    }

    void deleteStudent() {
        int rollNumber;
        cout << "Enter roll number to delete: ";
        cin >> rollNumber;
        studentList.deleteStudent(rollNumber);
    }

    void searchStudent() {
        int rollNumber;
        cout << "Enter roll number to search: ";
        cin >> rollNumber;
        Student *student = studentList.searchStudent(rollNumber);
        if (student) {
            cout << "Name: " << student->name << ", Roll Number: " << student-
>rollNumber
                 << ", Department: " << student->department << ", Section: "
<< student->section
                 << ", CGPA: " << student->cgpa << endl;
        } else {
            cout << "Student not found!" << endl;
        }
    }

    void updateStudent() {
        int rollNumber;
        string name, department, section;
        float cgpa;
        cout << "Enter roll number to update: ";
        cin >> rollNumber;
        cout << "Enter new name: ";
        cin >> name;
        cout << "Enter new department: ";
        cin >> department;
        cout << "Enter new section: ";
        cin >> section;
        cout << "Enter new CGPA: ";
        cin >> cgpa;

        studentList.updateStudent(rollNumber, name, department, section,
cgpa);
    }

    void displayAllStudents() {
        studentList.displayAllStudents();
    }

    void sortStudentsByCGPA() {
        studentList.sortStudentsByCGPA();
    }

    void sortStudentsByName() {
        studentList.sortStudentsByName();
    }

    void saveToFile() {
        ofstream outFile("students.txt");
        if (!outFile) {
            cerr << "File could not be opened for writing!" << endl;
            return;
        }

        Student *temp = studentList.head;
        while (temp != nullptr) {
            outFile << temp->name << " " << temp->rollNumber << " " << temp-
>department
                    << " " << temp->section << " " << temp->cgpa << endl;
            temp = temp->next;
        }

        outFile.close();
    }

    void loadFromFile() {
        ifstream inFile("students.txt");
        if (!inFile) {
            cerr << "File could not be opened for reading!" << endl;
            return;
        }

        string name, department, section;
        int rollNumber;
        float cgpa;
        while (inFile >> name >> rollNumber >> department >> section >> cgpa)
{
            studentList.addStudent(name, rollNumber, department, section,
cgpa);
        }

        inFile.close();
    }
};

int main() {
    StudentLMS lms;
    int choice;

    do {
        cout << "\nStudent LMS Menu\n";
        cout << "1. Add Student\n";
        cout << "2. Delete Student\n";
        cout << "3. Search Student\n";
        cout << "4. Update Student\n";
        cout << "5. Display All Students\n";
        cout << "6. Sort Students by CGPA\n";
        cout << "7. Sort Students by Name\n";
        cout << "8. Save to File\n";
        cout << "9. Load from File\n";
        cout << "10. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            lms.addStudent();
            break;
        case 2:
            lms.deleteStudent();
            break;
        case 3:
            lms.searchStudent();
            break;
        case 4:
            lms.updateStudent();
            break;
        case 5:
            lms.displayAllStudents();
            break;
        case 6:
            lms.sortStudentsByCGPA();
            break;
        case 7:
            lms.sortStudentsByName();
            break;
        case 8:
            lms.saveToFile();
            break;
        case 9:
            lms.loadFromFile();
            break;
        case 10:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice! Please try again." << endl;
        }
    } while (choice != 10);

    return 0;
}
```