**Project: Student Record Management System**

**Objectives**

1. *To understand and implement various data structures.*
2. *To apply algorithms for searching and sorting.*
3. *To practice procedural programming principles.*

**Features:**

1. Add a Student:

**-** Store student information: name, roll number, and grades.

**-** Use a linked list to store students for easy insertion and deletion.

1. Search for a Student:

**-** Search by roll number using linear search.

1. Delete a Student:

**-** Remove a student from the list using the roll number.

1. List All Students:

**-** List students in alphabetical order by name using a sorting algorithm.

1. Calculate Average Grades:

**-** Calculate and display the average grade of all students.

***Data Structures Used***

**-** Linked List: For storing and managing the list of students.

***Algorithms Used***

**-** Linear Search: For searching students by roll number.

**-** Sorting Algorithm: For sorting students by name (e.g., bubble sort or insertion sort).

**Implementation Steps:**

1. Define Structures:

**-** *Student*: Contains attributes like name, roll number, and grades.

**-** *Node*: Linked list node containing a *Student* and a pointer to the next node.

1. Implement Linked List:

**-** Functions for adding, deleting, and searching students in the list.

1. Implement Sorting:

**-** Sorting algorithm to list students in alphabetical order by name.

1. Develop UI:

**-** Create a simple command-line interface (CLI) to interact with the student record system.

1. Testing:

**-** Test each feature thoroughly with different scenarios to ensure correctness and efficiency.

Program:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Student

{

char name[100];

int rollNumber;

char grade[3]; // Grade as a string to handle letter grades like A+, B, etc.

} Student;

typedef struct Node

{

Student student;

struct Node \*next;

} Node;

Node \*head = NULL;

Node \*createNode(char \*name, int rollNumber, char \*grade)

{

Node \*newNode = (Node \*)malloc(sizeof(Node));

if (newNode == NULL)

{

perror("Error allocating memory");

exit(EXIT\_FAILURE);

}

strcpy(newNode->student.name, name);

newNode->student.rollNumber = rollNumber;

strcpy(newNode->student.grade, grade);

newNode->next = NULL;

return newNode;

}

void addStudent(char \*name, int rollNumber, char \*grade)

{

Node \*newNode = createNode(name, rollNumber, grade);

newNode->next = head;

head = newNode;

}

Node \*searchStudent(int rollNumber)

{

Node \*current = head;

while (current != NULL)

{

if (current->student.rollNumber == rollNumber)

{

return current;

}

current = current->next;

}

return NULL;

}

void deleteStudent(int rollNumber)

{

Node \*current = head;

Node \*previous = NULL;

while (current != NULL && current->student.rollNumber != rollNumber)

{

previous = current;

current = current->next;

}

if (current == NULL)

{

printf("Student with roll number %d not found.\n", rollNumber);

return;

}

if (previous == NULL)

{

head = current->next;

}

else

{

previous->next = current->next;

}

free(current);

printf("Student with roll number %d deleted.\n", rollNumber);

}

void swapNodes(Node \*a, Node \*b)

{

Student temp = a->student;

a->student = b->student;

b->student = temp;

}

void sortStudents()

{

if (head == NULL)

return;

int swapped;

Node \*ptr1;

Node \*lptr = NULL;

do

{

swapped = 0;

ptr1 = head;

while (ptr1->next != lptr)

{

if (strcmp(ptr1->student.name, ptr1->next->student.name) > 0)

{

swapNodes(ptr1, ptr1->next);

swapped = 1;

}

ptr1 = ptr1->next;

}

lptr = ptr1;

} while (swapped);

}

void listStudents()

{

sortStudents();

printf("\nAll Students Data:\n");

printf("--------------------------------------------------\n");

printf("Name | Roll Number | Grade\n");

printf("--------------------------------------------------\n");

Node \*current = head;

while (current != NULL)

{

printf("%-30s | %-11d | %s\n",

current->student.name, current->student.rollNumber, current->student.grade);

current = current->next;

}

printf("--------------------------------------------------\n");

}

void calculateAverageGrade()

{

if (head == NULL)

{

printf("No students in the list.\n");

return;

}

int count = 0;

int totalGradePoints = 0;

Node \*current = head;

while (current != NULL)

{

if (strcmp(current->student.grade, "A+") == 0)

totalGradePoints += 10;

else if (strcmp(current->student.grade, "A") == 0)

totalGradePoints += 9;

else if (strcmp(current->student.grade, "B+") == 0)

totalGradePoints += 8;

else if (strcmp(current->student.grade, "B") == 0)

totalGradePoints += 7;

else if (strcmp(current->student.grade, "C+") == 0)

totalGradePoints += 6;

else if (strcmp(current->student.grade, "C") == 0)

totalGradePoints += 5;

else if (strcmp(current->student.grade, "D+") == 0)

totalGradePoints += 4;

else if (strcmp(current->student.grade, "D") == 0)

totalGradePoints += 3;

else if (strcmp(current->student.grade, "E") == 0)

totalGradePoints += 2;

else if (strcmp(current->student.grade, "F") == 0)

totalGradePoints += 1;

count++;

current = current->next;

}

float averageGradePoints = (float)totalGradePoints / count;

printf("Average Grade Points: %.2f\n", averageGradePoints);

}

void saveToFile()

{

FILE \*file = fopen("students.txt", "w");

if (file == NULL)

{

perror("Error opening file for writing");

return;

}

Node \*current = head;

while (current != NULL)

{

fprintf(file, "%s\n%d\n%s\n", current->student.name, current->student.rollNumber, current->student.grade);

current = current->next;

}

fclose(file);

printf("Data saved to file.\n");

}

void loadFromFile()

{

FILE \*file = fopen("students.txt", "r");

if (file == NULL)

{

perror("Error opening file for reading");

return;

}

char name[100];

int rollNumber;

char grade[3];

while (fgets(name, sizeof(name), file))

{

name[strcspn(name, "\n")] = 0; // Remove newline character

if (fscanf(file, "%d\n", &rollNumber) != 1)

break; // Read roll number

if (fgets(grade, sizeof(grade), file) == NULL)

break; // Read grade

grade[strcspn(grade, "\n")] = 0; // Remove newline character

addStudent(name, rollNumber, grade);

}

fclose(file);

printf("Data loaded from file.\n");

}

int main()

{

int choice;

char name[100];

int rollNumber;

char grade[3];

loadFromFile(); // Load data from file when the program starts

while (1)

{

printf("\n1. Add Student\n");

printf("2. Search Student by Roll Number\n");

printf("3. Delete Student by Roll Number\n");

printf("4. List All Students\n");

printf("5. Calculate Average Grade\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar(); // consume newline

switch (choice)

{

case 1:

printf("Enter name: ");

fgets(name, sizeof(name), stdin);

name[strcspn(name, "\n")] = 0; // Remove newline character

printf("Enter roll number: ");

scanf("%d", &rollNumber);

getchar(); // consume newline

printf("Enter grade (A+, A, B+, B, C+, C, D+, D, E, F): ");

fgets(grade, sizeof(grade), stdin);

grade[strcspn(grade, "\n")] = 0; // Remove newline character

addStudent(name, rollNumber, grade);

break;

case 2:

printf("Enter roll number: ");

scanf("%d", &rollNumber);

getchar(); // consume newline

Node \*student = searchStudent(rollNumber);

if (student)

{

printf("Name: %s, Roll Number: %d, Grade: %s\n",

student->student.name, student->student.rollNumber, student->student.grade);

}

else

{

printf("Student not found.\n");

}

break;

case 3:

printf("Enter roll number: ");

scanf("%d", &rollNumber);

getchar(); // consume newline

deleteStudent(rollNumber);

break;

case 4:

listStudents();

break;

case 5:

calculateAverageGrade();

break;

case 6:

saveToFile(); // Save data to file when the program exits

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}