#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Structure for student information

typedef struct student {

char name[50];

int id;

float grade;

struct student \*next;

} Student;

Student \*head = NULL;

// Function to create a new student node

Student\* createStudent(const char\* name, int id, float grade) {

Student \*newStudent = (Student\*) malloc(sizeof(Student));

if (newStudent == NULL) {

perror("Memory allocation failed");

return NULL;

}

// Copy the student's name

strncpy(newStudent->name, name, sizeof(newStudent->name) - 1);

newStudent->name[sizeof(newStudent->name) - 1] = '\0';

newStudent->id = id;

newStudent->grade = grade;

newStudent->next = NULL;

return newStudent;

}

// Function to insert a student record into the linked list

void insertStudent(const char\* name, int id, float grade) {

Student \*newStudent = createStudent(name, id, grade);

if (head == NULL) {

head = newStudent; // If the list is empty, set as the head node

} else {

Student \*current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newStudent;

}

}

// Function to display all student records

void displayStudents() {

Student \*current = head;

if (current == NULL) {

printf("The linked list is empty.\n");

return;

}

while (current != NULL) {

printf("Name: %s, ID: %d, Grade: %.2f\n", current->name, current->id, current->grade);

current = current->next;

}

}

// Function to search for a student record by ID

Student\* searchStudentByID(int id) {

Student \*current = head;

while (current != NULL) {

if (current->id == id) {

return current;

}

current = current->next;

}

return NULL;

}

// Function to free the memory of the entire linked list

void freeMemory() {

Student \*current = head;

Student \*next;

while (current != NULL) {

next = current->next;

free(current);

current = next;

}

head = NULL;

printf("Memory has been freed.\n");

}

// Function to sort student records by grade (ascending order) using direct node swapping

void sortStudentsByGrade() {

if (head == NULL || head->next == NULL) {

return; // If the list is empty or only contains one student, no need to sort

}

Student \*i, \*j;

for (i = head; i != NULL; i = i->next) {

for (j = i->next; j != NULL; j = j->next) {

if (i->grade > j->grade) {

// Swap the data between nodes i and j

char tempName[50];

int tempID;

float tempGrade;

// Swap names

strcpy(tempName, i->name);

strcpy(i->name, j->name);

strcpy(j->name, tempName);

// Swap IDs

tempID = i->id;

i->id = j->id;

j->id = tempID;

// Swap grades

tempGrade = i->grade;

i->grade = j->grade;

j->grade = tempGrade;

}

}

}

}

int main() {

int choice;

char name[50];

int id;

float grade;

do {

printf("\n1. Insert a student record\n");

printf("2. Display student records\n");

printf("3. Sort records by grade\n");

printf("4. Search for a record by ID\n");

printf("5. Delete the list\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar();

switch (choice) {

case 1:

printf("Enter Name: ");

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

name[strcspn(name, "\n")] = '\0';

printf("Enter ID: ");

scanf("%d", &id);

printf("Enter Grade: ");

scanf("%f", &grade);

insertStudent(name, id, grade);

break;

case 2:

displayStudents();

break;

case 3:

sortStudentsByGrade();

break;

case 4:

printf("Enter ID: ");

scanf("%d", &id);

Student \*found = searchStudentByID(id);

if (found != NULL) {

printf("Student found: Name: %s, ID: %d, Grade: %.2f\n", found->name, found->id, found->grade);

} else {

printf("No student found with this ID.\n");

}

break;

case 5:

freeMemory();

break;

case 6:

printf("Exiting program...\n");

break;

default:

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

}

} while (choice != 6);

return 0;

}