#include <iostream>

#include <vector>

#include <fstream>

#include <algorithm>

#include <string>

using namespace std;

struct Course {

int course\_id;

string course\_name;

string instructor;

int credits;

};

vector<Course> courses;

const string DATA\_FILE = "courses\_data.txt";

void loadCourses() {

ifstream file(DATA\_FILE);

string line;

while (getline(file, line)) {

size\_t pos1 = line.find(",");

size\_t pos2 = line.find(",", pos1 + 1);

size\_t pos3 = line.find(",", pos2 + 1);

int id = stoi(line.substr(0, pos1));

string name = line.substr(pos1 + 1, pos2 - pos1 - 1);

string instructor = line.substr(pos2 + 1, pos3 - pos2 - 1);

int credits = stoi(line.substr(pos3 + 1));

courses.push\_back({id, name, instructor, credits});

}

file.close();

}

void storeCourses() {

ofstream file(DATA\_FILE);

for (const auto& course : courses) {

file << course.course\_id << "," << course.course\_name << ","

<< course.instructor << "," << course.credits << endl;

}

file.close();

}

void addCourse(int id, string name, string instructor, int credits) {

courses.push\_back({id, name, instructor, credits});

storeCourses();

}

void updateCourse(int id, string newName = "", string newInstructor = "", int newCredits = -1) {

for (auto& course : courses) {

if (course.course\_id == id) {

if (!newName.empty()) course.course\_name = newName;

if (!newInstructor.empty()) course.instructor = newInstructor;

if (newCredits != -1) course.credits = newCredits;

storeCourses();

return;

}

}

cout << "Course ID not found!" << endl;

}

void deleteCourse(int id) {

courses.erase(remove\_if(courses.begin(), courses.end(),

[id](Course& course) { return course.course\_id == id; }),

courses.end());

storeCourses();

}

vector<Course> linearSearch(const string& keyword) {

vector<Course> results;

for (const auto& course : courses) {

if (course.course\_name.find(keyword) != string::npos || course.instructor.find(keyword) != string::npos) {

results.push\_back(course);

}

}

return results;

}

int binarySearch(const vector<Course>& sortedCourses, const string& target, const string& key) {

int low = 0, high = sortedCourses.size() - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if ((key == "course\_name" && sortedCourses[mid].course\_name == target) ||

(key == "instructor" && sortedCourses[mid].instructor == target)) {

return mid;

} else if ((key == "course\_name" && sortedCourses[mid].course\_name < target) ||

(key == "instructor" && sortedCourses[mid].instructor < target)) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return -1;

}

void bubbleSort(vector<Course>& data, const string& key) {

for (size\_t i = 0; i < data.size() - 1; i++) {

for (size\_t j = 0; j < data.size() - i - 1; j++) {

if ((key == "course\_name" && data[j].course\_name > data[j + 1].course\_name) ||

(key == "credits" && data[j].credits > data[j + 1].credits)) {

swap(data[j], data[j + 1]);

}

}

}

}

void selectionSort(vector<Course>& data, const string& key) {

for (size\_t i = 0; i < data.size() - 1; i++) {

size\_t minIndex = i;

for (size\_t j = i + 1; j < data.size(); j++) {

if ((key == "course\_name" && data[j].course\_name < data[minIndex].course\_name) ||

(key == "credits" && data[j].credits < data[minIndex].credits)) {

minIndex = j;

}

}

swap(data[i], data[minIndex]);

}

}

void displayCourses(const vector<Course>& data) {

for (const auto& course : data) {

cout << "ID: " << course.course\_id << ", Name: " << course.course\_name

<< ", Instructor: " << course.instructor

<< ", Credits: " << course.credits << endl;

}

}

int main() {

loadCourses();

int choice;

do {

cout << "\n=== Course Management System ===\n";

cout << "1. Add a Course\n";

cout << "2. Update a Course\n";

cout << "3. Delete a Course\n";

cout << "4. Display All Courses\n";

cout << "5. Search for a Course (Linear Search)\n";

cout << "6. Search for a Course (Binary Search)\n";

cout << "7. Sort Courses\n";

cout << "0. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

int id, credits;

string name, instructor;

cout << "Enter Course ID: ";

cin >> id;

cin.ignore();

cout << "Enter Course Name: ";

getline(cin, name);

cout << "Enter Instructor: ";

getline(cin, instructor);

cout << "Enter Credits: ";

cin >> credits;

addCourse(id, name, instructor, credits);

break;

}

case 2: {

int id, newCredits;

string newName, newInstructor;

cout << "Enter Course ID to Update: ";

cin >> id;

cin.ignore();

cout << "Enter New Name (leave blank to skip): ";

getline(cin, newName);

cout << "Enter New Instructor (leave blank to skip): ";

getline(cin, newInstructor);

cout << "Enter New Credits (-1 to skip): ";

cin >> newCredits;

updateCourse(id, newName, newInstructor, newCredits);

break;

}

case 3: {

int id;

cout << "Enter Course ID to Delete: ";

cin >> id;

deleteCourse(id);

break;

}

case 4:

displayCourses(courses);

break;

case 5: {

string keyword;

cout << "Enter Keyword for Linear Search: ";

cin.ignore();

getline(cin, keyword);

vector<Course> results = linearSearch(keyword);

if (!results.empty()) {

displayCourses(results);

} else {

cout << "No courses found!" << endl;

}

break;

}

case 6: {

string key, target;

cout << "Search by (course\_name/instructor): ";

cin >> key;

cin.ignore();

cout << "Enter Target Value: ";

getline(cin, target);

bubbleSort(courses, key);

int index = binarySearch(courses, target, key);

if (index != -1) {

displayCourses({courses[index]});

} else {

cout << "Course not found!" << endl;

}

break;

}

case 7: {

string key;

cout << "Sort by (course\_name/credits): ";

cin >> key;

selectionSort(courses, key);

cout << "\nCourses After Sorting:\n";

displayCourses(courses);

break;

}

case 0:

cout << "Exiting program. Goodbye!\n";

break;

default:

cout << "Invalid choice! Please try again.\n";

}

} while (choice != 0);

return 0;

}