**Documentation**

**University:**Salim Habib University

**Project:**

University Management System

**Members:**

* Omaima sarfaraz
* Mariyam M.Salman

**Semester/Section**:

* fall semester
* SE
* Batch:1

**Introduction:**

This program implements a **University Management System** that performs basic operations related to managing students, courses, and attendance records. It is designed to be interactive, user-friendly, and modular, allowing for easy extension and maintenance.

### **Key Features:**

1. **User Authentication**:
   1. The program ensures security by requiring login credentials before accessing the system.
   2. It uses a predefined username-password pair (admin, admin123) for authentication.
2. **Student Management**:
   1. **Add Student**: Users can add a new student by providing an ID, name, and department.
   2. **View Students**: Displays the list of all students with their details.
3. **Course Management**:
   1. **Add Course**: Users can create a new course by entering a course ID, course name, and instructor's name.
   2. **View Courses**: Displays all registered courses with their details.
4. **Attendance Management**:
   1. **Mark Attendance**: Allows marking attendance for a specific course and student.
   2. **View Attendance**: Displays attendance records for a particular course.
5. **Persistent Data Storage**:
   1. The program saves student and course data to text files (students.txt and courses.txt) and reloads them on startup, ensuring data persistence across sessions.

**Tools and Technologies:**

Dev c++

**Project Design:**

### **Pseudocode for University Management System:**

**Start**

1. **Load Data**
   1. Load students from students.txt.
   2. Load courses from courses.txt.
2. **Login System**
   1. Prompt user for username and password.
   2. If valid credentials:
      1. Print "Login successful."
   3. Else:
      1. Print "Invalid username or password."
      2. Exit program.
3. **Main Menu**
   1. Repeat until the user chooses to exit:
      1. Display menu options:
         1. 1: Add Student
         2. 2: View Students
         3. 3: Add Course
         4. 4: View Courses
         5. 5: Mark Attendance
         6. 6: View Attendance
         7. 7: Save and Exit
      2. Get user choice.
4. **Perform Action Based on Choice**
   1. **Option 1: Add Student**
      1. Input student ID, name, and department.
      2. Add to the students array.
   2. **Option 2: View Students**
      1. Display all students and their details.
   3. **Option 3: Add Course**
      1. Input course ID, name, and instructor.
      2. Add to the courses array.
   4. **Option 4: View Courses**
      1. Display all courses and their details.
   5. **Option 5: Mark Attendance**
      1. Input course ID and find it in the courses array.
      2. Input student ID and find it in the students array.
      3. Input attendance status (1 for present, 0 for absent).
      4. Record attendance in the attendance array.
   6. **Option 6: View Attendance**
      1. Input course ID and find it in the courses array.
      2. Display attendance for all students in the course.
   7. **Option 7: Save and Exit**
      1. Save students to students.txt.
      2. Save courses to courses.txt.
      3. Print "Data saved successfully."
      4. Exit program.

**Code:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

// User authentication

struct User {

string username;

string password;

};

User users[] = { {"admin", "admin123"} };

bool login() {

string username, password;

cout << "\n--- Login ---\n";

cout << "Username: ";

cin >> username;

cout << "Password: ";

cin >> password;

for (int i = 0; i < sizeof(users) / sizeof(users[0]); ++i) {

if (users[i].username == username && users[i].password == password) {

cout << "Login successful!\n";

return true;

}

}

cout << "Invalid username or password.\n";

return false;

}

struct Student {

int id;

string name;

string department;

};

// Course data

struct Course {

int courseId;

string courseName;

string instructor;

};

// Attendance data

struct Attendance {

int studentId;

bool isPresent; // true for present, false for absent

};

Student students[100];

Course courses[100];

Attendance attendance[100][100]; // Assuming a max of 100 students and 100 courses

int studentCount = 0;

int courseCount = 0;

void displayStudent(const Student& student) {

cout << "ID: " << student.id << ", Name: " << student.name << ", Department: " << student.department << endl;

}

void displayCourse(const Course& course) {

cout << "Course ID: " << course.courseId << ", Course Name: " << course.courseName << ", Instructor: " << course.instructor << endl;

}

void saveStudentsToFile() {

ofstream outFile("students.txt");

if (!outFile) {

cout << "Error: Unable to save students to file.\n";

return;

}

for (int i = 0; i < studentCount; ++i) {

outFile << students[i].id << "\n" << students[i].name << "\n" << students[i].department << "\n";

}

outFile.close();

}

void loadStudentsFromFile() {

ifstream inFile("students.txt");

if (!inFile) {

cout << "Note: No existing student data found. Starting fresh.\n";

return;

}

studentCount = 0;

while (inFile) {

Student s;

inFile >> s.id;

inFile.ignore(); // Ignore newline after reading id

if (inFile.eof()) break;

getline(inFile, s.name);

getline(inFile, s.department);

students[studentCount++] = s;

}

inFile.close();

}

void saveCoursesToFile() {

ofstream outFile("courses.txt");

if (!outFile) {

cout << "Error: Unable to save courses to file.\n";

return;

}

for (int i = 0; i < courseCount; ++i) {

outFile << courses[i].courseId << "\n" << courses[i].courseName << "\n" << courses[i].instructor << "\n";

}

outFile.close();

}

void loadCoursesFromFile() {

ifstream inFile("courses.txt");

if (!inFile) {

cout << "Note: No existing course data found. Starting fresh.\n";

return;

}

courseCount = 0;

while (inFile) {

Course c;

inFile >> c.courseId;

inFile.ignore(); // Ignore newline after reading courseId

if (inFile.eof()) break;

getline(inFile, c.courseName);

getline(inFile, c.instructor);

courses[courseCount++] = c;

}

inFile.close();

}

void markAttendance() {

int courseId, studentId, status;

cout << "Enter course ID to mark attendance: ";

cin >> courseId;

// Find the course by ID

int courseIndex = -1;

for (int i = 0; i < courseCount; ++i) {

if (courses[i].courseId == courseId) {

courseIndex = i;

break;

}

}

if (courseIndex == -1) {

cout << "Invalid course ID.\n";

return;

}

cout << "Enter student ID to mark attendance: ";

cin >> studentId;

// Find the student by ID

int studentIndex = -1;

for (int i = 0; i < studentCount; ++i) {

if (students[i].id == studentId) {

studentIndex = i;

break;

}

}

if (studentIndex == -1) {

cout << "Invalid student ID.\n";

return;

}

cout << "Enter attendance status (1 for present, 0 for absent): ";

cin >> status;

// Mark attendance

attendance[courseIndex][studentIndex].studentId = studentId;

attendance[courseIndex][studentIndex].isPresent = (status == 1);

cout << "Attendance for student " << students[studentIndex].name

<< " has been marked as "

<< (status == 1 ? "Present" : "Absent") << ".\n";

}

void viewAttendance() {

int courseId;

cout << "Enter course ID to view attendance: ";

cin >> courseId;

// Find the course by ID

int courseIndex = -1;

for (int i = 0; i < courseCount; ++i) {

if (courses[i].courseId == courseId) {

courseIndex = i;

break;

}

}

if (courseIndex == -1) {

cout << "Invalid course ID.\n";

return;

}

cout << "\nAttendance for Course: " << courses[courseIndex].courseName << "\n";

bool attendanceFound = false;

// Loop through all students and check attendance

for (int i = 0; i < studentCount; ++i) {

if (attendance[courseIndex][i].studentId == students[i].id) {

cout << students[i].name << " - "

<< (attendance[courseIndex][i].isPresent ? "Present" : "Absent") << endl;

attendanceFound = true;

}

}

if (!attendanceFound) {

cout << "No attendance records found for this course.\n";

}

}

int main() {

loadStudentsFromFile();

loadCoursesFromFile();

if (!login()) {

return 0;

}

int choice;

do {

cout << "\nUniversity Management System\n";

cout << "1. Add Student\n";

cout << "2. View Students\n";

cout << "3. Add Course\n";

cout << "4. View Courses\n";

cout << "5. Mark Attendance\n";

cout << "6. View Attendance\n";

cout << "7. Save and Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (cin.fail()) { // Check for invalid input

cin.clear();

cin.ignore(1000, '\n');

cout << "Invalid choice. Please enter a number.\n";

continue;

}

switch (choice) {

case 1: {

int id;

string name, department;

cout << "Enter student ID: ";

cin >> id;

cin.ignore();

cout << "Enter student name: ";

getline(cin, name);

cout << "Enter student department: ";

getline(cin, department);

students[studentCount].id = id;

students[studentCount].name = name;

students[studentCount].department = department;

++studentCount;

break;

}

case 2: {

cout << "\nList of Students:\n";

for (int i = 0; i < studentCount; ++i) {

displayStudent(students[i]);

}

break;

}

case 3: {

int courseId;

string courseName, instructor;

cout << "Enter course ID: ";

cin >> courseId;

cin.ignore();

cout << "Enter course name: ";

getline(cin, courseName);

cout << "Enter instructor name: ";

getline(cin, instructor);

courses[courseCount].courseId = courseId;

courses[courseCount].courseName = courseName;

courses[courseCount].instructor = instructor;

++courseCount;

break;

}

case 4: {

cout << "\nList of Courses:\n";

for (int i = 0; i < courseCount; ++i) {

displayCourse(courses[i]);

}

break;

}

case 5: {

markAttendance();

break;

}

case 6: {

viewAttendance();

break;

}

case 7: {

saveStudentsToFile();

saveCoursesToFile();

cout << "Data saved successfully. Exiting..." << endl;

break;

}

default:

cout << "Invalid choice. Please try again." << endl;

}

} while (choice != 7);

return 0;

}

**University Management System** designed to handle students, courses, and attendance. Here's how it works:

1. **Initialization**:
   1. The program starts by **loading data** about students and courses from external files (students.txt and courses.txt).
   2. It defines key structures:
      1. Student (ID, name, department).
      2. Course (ID, name, instructor).
      3. Attendance (track whether a student is present for a course).
2. **User Authentication**:
   1. The program requires **login credentials** (username/password) to proceed. Only pre-defined users are allowed.
3. **Main Menu**:
   1. After a successful login, the user is presented with a menu of options to manage students, courses, and attendance.
4. **Key Functionalities**:
   1. **Add Student**: Allows the user to input details of a new student and add them to the system.
   2. **View Students**: Displays a list of all students with their details.
   3. **Add Course**: Allows the user to add a new course with details (ID, name, instructor).
   4. **View Courses**: Displays a list of all available courses.
   5. **Mark Attendance**: Lets the user mark a student’s attendance (present/absent) for a specific course.
   6. **View Attendance**: Displays attendance records for a selected course.
   7. **Save and Exit**: Saves updated student and course data back to their respective files and exits the program.
5. **Data Persistence**:
   1. Student and course information is saved to files when changes are made. This ensures that data remains available for the next session.
6. **Input Validation**:
   1. The program includes basic checks for invalid inputs, such as wrong student/course IDs or invalid menu choices.

In short, the program enables basic management of a university's records, focusing on students, courses, and attendance tracking.

**Main Programming concepts:**

* Loops
* Arrays
* Structures
* Exceptional handlings
* Conditions

### **Test Case 1: Adding a Student**

**Input**:

1. Login with valid credentials (admin, admin123).
2. Choose option 1 (Add Student).
   1. Enter:
      1. Student ID: 101
      2. Name: Alice Johnson
      3. Department: Computer Science.
3. Choose option 2 (View Students).

**Expected Output**:

List of Students:  
ID: 101, Name: Alice Johnson, Department: Computer Science

### **Test Case 2: Adding a Course**

**Input**:

1. Login with valid credentials.
2. Choose option 3 (Add Course).
   1. Enter:
      1. Course ID: 501
      2. Course Name: Data Structures
      3. Instructor: Dr. Smith.
3. Choose option 4 (View Courses).

**Expected Output**:

List of Courses:  
Course ID: 501, Course Name: Data Structures, Instructor: Dr. Smith

### **Test Case 3: Marking Attendance**

**Preconditions**:

* A student with ID 101 and name Alice Johnson exists.
* A course with ID 501 and name Data Structures exists.

**Input**:

1. Login with valid credentials.
2. Choose option 5 (Mark Attendance).
   1. Enter:
      1. Course ID: 501
      2. Student ID: 101
      3. Attendance Status: 1 (Present).
3. Choose option 6 (View Attendance).
   1. Enter: Course ID: 501.

**Expected Output**:

Attendance for Course: Data Structures  
Alice Johnson - Present

### **Test Case 4: Invalid Student ID for Attendance**

**Input**:

1. Login with valid credentials.
2. Choose option 5 (Mark Attendance).
   1. Enter:
      1. Course ID: 501
      2. Student ID: 999
      3. Attendance Status: 1.

**Expected Output**:

Invalid student ID.

### **Test Case 5: Saving and Loading Data**

**Input**:

1. Add a student (ID: 102, Name: Bob Smith, Department: Physics).
2. Add a course (ID: 502, Name: Quantum Mechanics, Instructor: Dr. Brown).
3. Save and exit (option 7).
4. Restart the program and login.
5. Choose option 2 (View Students) and option 4 (View Courses).

**Expected Output After Restart**:

List of Students:  
ID: 102, Name: Bob Smith, Department: Physics  
  
List of Courses:  
Course ID: 502, Course Name: Quantum Mechanics, Instructor: Dr. Brown

### **Conclusion:**

This program provides a solid foundation for a University Management System with basic functionalities like managing students, courses, and attendance. While it meets the requirements for a small-scale system, it can be further enhanced to improve scalability, security, and usability. By addressing its limitations, this system could serve as a robust solution for educational institutions.