```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class Student
{
private:
   string name;
   int rollNumber;
   string branch;
   int semester;

public:
   Student(string name, int rollNumber, string branch, int semester)
      : name(name), rollNumber(rollNumber), branch(branch), semester(semester) {}

   string getName() const { return name; }
   int getRollNumber() const { return rollNumber; }
   string getBranch() const { return branch; }
   int getSemester() const { return semester; }

   void setName(const string &newName) { name = newName; }
   void setRollNumber(int newRollNumber) { rollNumber = newRollNumber; }
   void setBranch(const string &newBranch) { branch = newBranch; }
   void setSemester(int newSemester) { semester = newSemester; }

   void displayDetails() const
   {
      cout << "Name: " << name << endl;
      cout << "Roll Number: " << rollNumber << endl;
      cout << "Branch: " << branch << endl;
      cout << "Semester: " << semester << endl;
   }
};
class StudentManager
{
private:
   vector<Student> students;

public:
   void addStudent()
   {
      string name, branch;
      int rollNumber, semester;
      cout << "Enter Name: ";
```

```
    cin.ignore();
    getline(cin, name);
    cout << "Enter Roll Number: ";
    cin >> rollNumber;
    cout << "Enter Branch: ";
    cin.ignore();
    getline(cin, branch);
    cout << "Enter Semester: ";
    cin >> semester;
    students.push_back(Student(name, rollNumber, branch, semester));
    cout << "Student added successfully." << endl;
}
void deleteStudent()
{
    int rollNumber;
    cout << "Enter Roll Number of student to delete: ";
    cin >> rollNumber;
    auto it = find_if(
        students.begin(), students.end(),
        [rollNumber](const Student &student)
        { return student.getRollNumber() == rollNumber; });
    if (it != students.end())
    {
        students.erase(it);
        cout << "Student with roll number " << rollNumber << " deleted." << endl;
    }
    else
    {
        cout << "Student with roll number " << rollNumber << " not found." << endl;
    }
}
void editStudent()
{
    int rollNumber;
    cout << "Enter Roll Number of student to edit: ";
    cin >> rollNumber;
    auto it = find_if(
        students.begin(), students.end(),
        [rollNumber](const Student &student)
        { return student.getRollNumber() == rollNumber; });
    if (it != students.end())
    {
        string newName, newBranch;
        int newSemester;
        cout << "Enter New Name: ";
```

```
            cin.ignore();
            getline(cin, newName);
            cout << "Enter New Branch: ";
            getline(cin, newBranch);
            cout << "Enter New Semester: ";
            cin >> newSemester;
            it->setName(newName);
            it->setBranch(newBranch);
            it->setSemester(newSemester);
            cout << "Student details updated successfully." << endl;
        }
        else
        {
            cout << "Student with roll number " << rollNumber << " not found." << endl;
        }
    }
    void showAllStudents() const
    {
        if (students.empty())
        {
            cout << "No students found." << endl;
        }
        else
        {
            for (const auto &student : students)
            {
                student.displayDetails();
                cout << "-------------------------" << endl;
            }
        }
    }
};
class Paper
{
private:
    string name;
    int id;
    string date;
    int duration;

public:
    Paper(string name, int id, string date, int duration)
        : name(name), id(id), date(date), duration(duration) {}

    string getName() const { return name; }
```

```
    int getId() const { return id; }
    string getDate() const { return date; }
    int getDuration() const { return duration; }

    void setName(const string &newName) { name = newName; }
    void setId(int newId) { id = newId; }
    void setDate(const string &newDate) { date = newDate; }
    void setDuration(int newDuration) { duration = newDuration; }

    void displayDetails() const
    {
        cout << "Name: " << name << endl;
        cout << "ID: " << id << endl;
        cout << "Date: " << date << endl;
        cout << "Duration: " << duration << " minutes" << endl;
    }
    static bool compareByDate(const Paper &paper1, const Paper &paper2) { return paper1.date <
paper2.date; }
};
class PaperManager
{
private:
    vector<Paper> papers;

public:
    void addPaper()
    {
        string name, date;
        int id, duration;
        cout << "Enter Name: ";
        cin.ignore();
        getline(cin, name);
        cout << "Enter ID: ";
        cin >> id;
        cout << "Enter Date (YYYY-MM-DD): ";
        cin.ignore();
        getline(cin, date);
        cout << "Enter Duration (in minutes): ";
        cin >> duration;
        papers.push_back(Paper(name, id, date, duration));
        cout << "Paper added successfully." << endl;
    }
    void deletePaper()
    {
        int id;
```

```
    cout << "Enter ID of paper to delete: ";
    cin >> id;
    auto it = find_if(papers.begin(), papers.end(),
                      [id](const Paper &paper)
                      { return paper.getId() == id; });
    if (it != papers.end())
    {
        papers.erase(it);
        cout << "Paper with ID " << id << " deleted." << endl;
    }
    else
    {
        cout << "Paper with ID " << id << " not found." << endl;
    }
}
void editPaper()
{
    int id;
    cout << "Enter ID of paper to edit: ";
    cin >> id;
    auto it = find_if(papers.begin(), papers.end(),
                      [id](const Paper &paper)
                      { return paper.getId() == id; });
    if (it != papers.end())
    {
        string newName, newDate;
        int newDuration;
        cout << "Enter New Name: ";
        cin.ignore();
        getline(cin, newName);
        cout << "Enter New Date (YYYY-MM-DD): ";
        getline(cin, newDate);
        cout << "Enter New Duration (in minutes): ";
        cin >> newDuration;
        it->setName(newName);
        it->setDate(newDate);
        it->setDuration(newDuration);
        cout << "Paper details updated successfully." << endl;
    }
    else
    {
        cout << "Paper with ID " << id << " not found." << endl;
    }
}
void showAllPapersSortedByDate() const
```

```
    {
        vector<Paper> sortedPapers = papers;
        sort(sortedPapers.begin(), sortedPapers.end(), Paper::compareByDate);
        if (sortedPapers.empty())
        {
            cout << "No papers found." << endl;
        }
        else
        {
            for (const auto &paper : sortedPapers)
            {
                paper.displayDetails();
                cout << "-------------------------" << endl;
            }
        }
    }
};
class Course
{
private:
    string name;
    int courseId;
    string department;

public:
    Course(string name, int courseId, string department)
        : name(name), courseId(courseId), department(department) {}
    string getName() const { return name; }
    int getCourseId() const { return courseId; }
    string getDepartment() const { return department; }
    void setName(const string &newName) { name = newName; }
    void setCourseId(int newCourseId) { courseId = newCourseId; }
    void setDepartment(const string &newDepartment) { department = newDepartment; }
    void displayDetails() const
    {
        cout << "Name: " << name << endl;
        cout << "Course ID: " << courseId << endl;
        cout << "Department: " << department << endl;
    }
};
class CourseManager
{
private:
    vector<Course> courses;
```

```
public:
    void addCourse()
    {
        string name, department;
        int courseId;
        cout << "Enter Name: ";
        cin.ignore();
        getline(cin, name);
        cout << "Enter Course ID: ";
        cin >> courseId;
        cout << "Enter Department: ";
        cin.ignore();
        getline(cin, department);
        courses.push_back(Course(name, courseId, department));
        cout << "Course added successfully." << endl;
    }

    void deleteCourse()
    {
        int courseId;
        cout << "Enter Course ID of course to delete: ";
        cin >> courseId;
        auto it = find_if(courses.begin(), courses.end(),
                          [courseId](const Course &course)
                          { return course.getCourseId() == courseId; });
        if (it != courses.end())
        {
            courses.erase(it);
            cout << "Course with ID " << courseId << " deleted." << endl;
        }
        else
        {
            cout << "Course with ID " << courseId << " not found." << endl;
        }
    }

    void editCourse()
    {
        int courseId;
        cout << "Enter Course ID of course to edit: ";
        cin >> courseId;
        auto it = find_if(courses.begin(), courses.end(),
                          [courseId](const Course &course)
                          { return course.getCourseId() == courseId; });
        if (it != courses.end())
```

```
        {
            string newName, newDepartment;
            cout << "Enter New Name: ";
            cin.ignore();
            getline(cin, newName);
            cout << "Enter New Department: ";
            getline(cin, newDepartment);
            it->setName(newName);
            it->setDepartment(newDepartment);
            cout << "Course details updated successfully." << endl;
        }
        else
        {
            cout << "Course with ID " << courseId << " not found." << endl;
        }
    }

    void showAllCourses() const
    {
        if (courses.empty())
        {
            cout << "No courses found." << endl;
        }
        else
        {
            for (const auto &course : courses)
            {
                course.displayDetails();
                cout << "-------------------------" << endl;
            }
        }
    }
};
class Invigilator
{
private:
    string name;
    int id;
    string department;

public:
    Invigilator(string name, int id, string department)
        : name(name), id(id), department(department) {}

    string getName() const { return name; }
```

```
    int getId() const { return id; }
    string getDepartment() const { return department; }

    void setName(const string &newName) { name = newName; }
    void setId(int newId) { id = newId; }
    void setDepartment(const string &newDepartment) { department = newDepartment; }

    void displayDetails() const
    {
        cout << "Name: " << name << endl;
        cout << "ID: " << id << endl;
        cout << "Department: " << department << endl;
    }
};
class InvigilatorManager
{
private:
    vector<Invigilator> invigilators;

public:
    void addInvigilator()
    {
        string name, department;
        int id;
        cout << "Enter Name: ";
        cin.ignore();
        getline(cin, name);
        cout << "Enter ID: ";
        cin >> id;
        cout << "Enter Department: ";
        cin.ignore();
        getline(cin, department);
        invigilators.push_back(Invigilator(name, id, department));
        cout << "Invigilator added successfully." << endl;
    }

    void deleteInvigilator()
    {
        int id;
        cout << "Enter ID of invigilator to delete: ";
        cin >> id;
        auto it = find_if(invigilators.begin(), invigilators.end(),
                          [id](const Invigilator &invigilator)
                          { return invigilator.getId() == id; });
        if (it != invigilators.end())
```

```
    {
        invigilators.erase(it);
        cout << "Invigilator with ID " << id << " deleted." << endl;
    }
    else
    {
        cout << "Invigilator with ID " << id << " not found." << endl;
    }
}

void editInvigilator()
{
    int id;
    cout << "Enter ID of invigilator to edit: ";
    cin >> id;
    auto it = find_if(invigilators.begin(), invigilators.end(),
                      [id](const Invigilator &invigilator)
                      { return invigilator.getId() == id; });
    if (it != invigilators.end())
    {
        string newName, newDepartment;
        cout << "Enter New Name: ";
        cin.ignore();
        getline(cin, newName);
        cout << "Enter New Department: ";
        getline(cin, newDepartment);
        it->setName(newName);
        it->setDepartment(newDepartment);
        cout << "Invigilator details updated successfully." << endl;
    }
    else
    {
        cout << "Invigilator with ID " << id << " not found." << endl;
    }
}

void showAllInvigilators() const
{
    if (invigilators.empty())
    {
        cout << "No invigilators found." << endl;
    }
    else
    {
        for (const auto &invigilator : invigilators)
```

```
        {
            invigilator.displayDetails();
            cout << "-------------------------" << endl;
        }
    }
  }
};
class Result
{
private:
  int resultId;
  string studentName;
  int rollNumber;
  int marks;
  double percentage;

public:
  Result(int resultId, string studentName, int rollNumber, int marks)
    : resultId(resultId), studentName(studentName), rollNumber(rollNumber), marks(marks)
  {
    calculatePercentage();
  }

  int getResultId() const { return resultId; }
  string getStudentName() const { return studentName; }
  int getRollNumber() const { return rollNumber; }
  int getMarks() const { return marks; }
  double getPercentage() const { return percentage; }

  void setStudentName(const string &newName) { studentName = newName; }
  void setRollNumber(int newRollNumber) { rollNumber = newRollNumber; }
  void setMarks(int newMarks) { marks = newMarks; }

  void calculatePercentage()
  {
    percentage = (static_cast<double>(marks) / 100) * 100;
  }

  void displayDetails() const
  {
    cout << "Result ID: " << resultId << endl;
    cout << "Student Name: " << studentName << endl;
    cout << "Roll Number: " << rollNumber << endl;
    cout << "Marks: " << marks << endl;
    cout << "Percentage: " << percentage << "%" << endl;
```

```
    }
};
class ResultManager
{
private:
    vector<Result> results;
    int nextResultId = 1;

public:
    void addResult()
    {
        string studentName;
        int rollNumber, marks;
        cout << "Enter Student Name: ";
        cin.ignore();
        getline(cin, studentName);
        cout << "Enter Roll Number: ";
        cin >> rollNumber;
        cout << "Enter Marks: ";
        cin >> marks;
        results.push_back(Result(nextResultId++, studentName, rollNumber, marks));
        cout << "Result added successfully." << endl;
    }

    void deleteResult()
    {
        int resultId;
        cout << "Enter Result ID of result to delete: ";
        cin >> resultId;
        auto it = find_if(results.begin(), results.end(),
                    [resultId](const Result &result)
                    { return result.getResultId() == resultId; });
        if (it != results.end())
        {
            results.erase(it);
            cout << "Result with ID " << resultId << " deleted." << endl;
        }
        else
        {
            cout << "Result with ID " << resultId << " not found." << endl;
        }
    }

    void editResult()
    {
```

```
        int resultId;
        cout << "Enter Result ID of result to edit: ";
        cin >> resultId;
        auto it = find_if(results.begin(), results.end(),
                    [resultId](const Result &result)
                    { return result.getResultId() == resultId; });
        if (it != results.end())
        {
            string newName;
            int newRollNumber, newMarks;
            cout << "Enter New Student Name: ";
            cin.ignore();
            getline(cin, newName);
            cout << "Enter New Roll Number: ";
            cin >> newRollNumber;
            cout << "Enter New Marks: ";
            cin >> newMarks;
            it->setStudentName(newName);
            it->setRollNumber(newRollNumber);
            it->setMarks(newMarks);
            it->calculatePercentage();
            cout << "Result details updated successfully." << endl;
        }
        else
        {
            cout << "Result with ID " << resultId << " not found." << endl;
        }
    }

    void showAllResults() const
    {
        if (results.empty())
        {
            cout << "No results found." << endl;
        }
        else
        {
            for (const auto &result : results)
            {
                result.displayDetails();
                cout << "-------------------------" << endl;
            }
        }
    }
};
```

```
int main()
{
   StudentManager studentManager;
   PaperManager paperManager;
   CourseManager courseManager;
   InvigilatorManager invigilatorManager;
   ResultManager resultManager;
   int choice;

   do
   {
      cout << "\nMenu:\n"
          << "1. Student Management\n"
          << "2. Paper Management\n"
          << "3. Course Management\n"
          << "4. Invigilator Management\n"
          << "5. Result Management\n"
          << "6. Exit\n"
          << "Enter your choice: ";
      cin >> choice;

      switch (choice)
      {
      case 1:
      {
         int studentChoice;
         do
         {
            cout << "\nStudent Management Menu:\n"
                << "1. Add Student\n"
                << "2. Delete Student\n"
                << "3. Edit Student\n"
                << "4. Show All Students\n"
                << "5. Back to Main Menu\n"
                << "Enter your choice: ";
            cin >> studentChoice;

            switch (studentChoice)
            {
            case 1:
               studentManager.addStudent();
               break;
            case 2:
               studentManager.deleteStudent();
```

```
            break;
        case 3:
            studentManager.editStudent();
            break;
        case 4:
            studentManager.showAllStudents();
            break;
        case 5:
            cout << "Returning to Main Menu.\n";
            break;
        default:
            cout << "Invalid choice! Please try again.\n";
            break;
        }
    } while (studentChoice != 5);
    break;
}
case 2:
{
    int paperChoice;
    do
    {
        cout << "\nPaper Management Menu:\n"
             << "1. Add Paper\n"
             << "2. Delete Paper\n"
             << "3. Edit Paper\n"
             << "4. Show All Papers\n"
             << "5. Back to Main Menu\n"
             << "Enter your choice: ";
        cin >> paperChoice;

        switch (paperChoice)
        {
        case 1:
            paperManager.addPaper();
            break;
        case 2:
            paperManager.deletePaper();
            break;
        case 3:
            paperManager.editPaper();
            break;
        case 4:
            paperManager.showAllPapersSortedByDate();
            break;
```

```
        case 5:
            cout << "Returning to Main Menu.\n";
            break;
        default:
            cout << "Invalid choice! Please try again.\n";
            break;
        }
    } while (paperChoice != 5);
    break;
}
case 3:
{
    int courseChoice;
    do
    {
        cout << "\nPaper Management Menu:\n"
             << "1. Add Course\n"
             << "2. Delete Course\n"
             << "3. Edit Course\n"
             << "4. Show All Courses\n"
             << "5. Back to Main Menu\n"
             << "Enter your choice: ";
        cin >> courseChoice;

        switch (courseChoice)
        {
        case 1:
            courseManager.addCourse();
            break;
        case 2:
            courseManager.deleteCourse();
            break;
        case 3:
            courseManager.editCourse();
            break;
        case 4:
            courseManager.showAllCourses();
            break;
        case 5:
            cout << "Returning to Main Menu.\n";
            break;
        default:
            cout << "Invalid choice! Please try again.\n";
            break;
        }
```

```
    } while (courseChoice != 5);
    break;
}
case 4:
{
    int inviChoice;
    do
    {
        cout << "\nPaper Management Menu:\n"
             << "1. Add Invigilator\n"
             << "2. Delete Invigilator\n"
             << "3. Edit Invigilator\n"
             << "4. Show All Invigilator\n"
             << "5. Back to Main Menu\n"
             << "Enter your choice: ";
        cin >> inviChoice;

        switch (inviChoice)
        {
        case 1:
            invigilatorManager.addInvigilator();
            break;
        case 2:
            invigilatorManager.deleteInvigilator();
            break;
        case 3:
            invigilatorManager.editInvigilator();
            break;
        case 4:
            invigilatorManager.showAllInvigilators();
            break;
        case 5:
            cout << "Returning to Main Menu.\n";
            break;
        default:
            cout << "Invalid choice! Please try again.\n";
            break;
        }
    } while (inviChoice != 5);
    break;
}
case 5:
{
    int resChoice;
    do
```

```
        {
            cout << "\nPaper Management Menu:\n"
                 << "1. Add Result\n"
                 << "2. Delete Result\n"
                 << "3. Edit Result\n"
                 << "4. Show All Result\n"
                 << "5. Back to Main Menu\n"
                 << "Enter your choice: ";
            cin >> resChoice;

            switch (resChoice)
            {
            case 1:
                resultManager.addResult();
                break;
            case 2:
                resultManager.deleteResult();
                break;
            case 3:
                resultManager.editResult();
                break;
            case 4:
                resultManager.showAllResults();
                break;
            case 5:
                cout << "Returning to Main Menu.\n";
                break;
            default:
                cout << "Invalid choice! Please try again.\n";
                break;
            }
        } while (resChoice != 5);
        break;
    }

    // Similar cases for other options
    case 6:
        cout << "Exiting program. Goodbye!\n";
        break;
    default:
        cout << "Invalid choice! Please try again.\n";
        break;
    }
} while (choice != 6);
```

```
    return 0;
}
```