main.cpp

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
using namespace std;

// Define the Record classes
class Course {
public:
    string courseID;
    string courseName;
    int credits;

    Course(string id, string name, int cr) : courseID(id), courseName(name), credits(cr) {}
};

class Student {
public:
    string studentID;
    string studentName;

    Student(string id, string name) : studentID(id), studentName(name) {}
};

class Teacher {
public:
    string teacherID;
    string teacherName;

    Teacher(string id, string name) : teacherID(id), teacherName(name) {}
};

class Enrollment {
public:
    string studentID;
    string courseID;

    Enrollment(string sid, string cid) : studentID(sid), courseID(cid) {}
};

// Data storage
vector<Student> students;
vector<Teacher> teachers;
vector<Course> courses;
vector<Enrollment> enrollments;

// Utility functions
void listStudents() {
    cout << "\nList of Students:" << endl;
    for (const auto& student : students) {
        cout << "ID: " << student.studentID << ", Name: " << student.studentName << endl;
    }
}

void listTeachers() {
    cout << "\nList of Teachers:" << endl;
    for (const auto& teacher : teachers) {
        cout << "ID: " << teacher.teacherID << ", Name: " << teacher.teacherName << endl;
    }
}

void listCourses() {
    cout << "\nList of Courses:" << endl;
    for (const auto& course : courses) {
        cout << "ID: " << course.courseID << ", Name: " << course.courseName << ", Credits: " << course.credits << endl;
    }
}

void listEnrollments() {
    cout << "\nEnrollments by Student:" << endl;
    unordered_map<string, vector<string>> studentCourses;

    // Organize enrollments by student
    for (const auto& enrollment : enrollments) {
        studentCourses[enrollment.studentID].push_back(enrollment.courseID);
    }

    // Display enrollments grouped by student
    for (const auto& student : students) {
        cout << "Student ID: " << student.studentID << ", Name: " << student.studentName << endl;
        if (studentCourses.count(student.studentID)) {
            for (const auto& courseID : studentCourses[student.studentID]) {
                auto it = find_if(courses.begin(), courses.end(), [&](const Course& c) { return c.courseID == courseID; });
                if (it != courses.end()) {
                    cout << "  - Course ID: " << it->courseID << ", Name: " << it->courseName << ", Credits: " << it->credits << endl;
                }
            }
        } else {
            cout << "  No courses enrolled." << endl;
        }
    }
}

void showMenu() {
    cout << "\nApplication Menu:" << endl;
    cout << "1. List Students" << endl;
    cout << "2. List Teachers" << endl;
    cout << "3. List Courses" << endl;
    cout << "4. List Enrollments" << endl;
    cout << "5. Exit" << endl;
}

int main() {
    // Sample data
    students.push_back(Student("S001", "Alice"));
    students.push_back(Student("S002", "Bob"));

    teachers.push_back(Teacher("T001", "Dr. Smith"));
    teachers.push_back(Teacher("T002", "Prof. Johnson"));

    courses.push_back(Course("C001", "Mathematics", 3));
    courses.push_back(Course("C002", "Physics", 4));

    enrollments.push_back(Enrollment("S001", "C001"));
    enrollments.push_back(Enrollment("S001", "C002"));
    enrollments.push_back(Enrollment("S002", "C001"));

    int choice;
    do {
        showMenu();
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                listStudents();
                break;
            case 2:
                listTeachers();
                break;
            case 3:
                listCourses();
                break;
            case 4:
                listEnrollments();
                break;
            case 5:
                cout << "Exiting the application." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    } while (choice != 5);

    return 0;
```

```
}

```