Building a Course Registration System in C :-

Managing course registrations for students can be a tedious process, but programming offers a way to automate and simplify the task. In this blog, we'll walk through a Course Registration System implemented in the C programming language.

Overview :

This program demonstrates a simple command-line application to:

1. Add students.
2. Add courses.
3. Enroll students in courses.
4. View students and courses.

The system is built using **structs** to represent students and courses, with global arrays to store them.

Key Features of the System :

1. **Add Student**: Enables adding a student with an ID and name.
2. **Add Course**: Allows adding a course with an ID, name, and schedule.
3. **Enroll Student in a Course**: Associates a student with a course.
4. **View Students**: Displays all added students.
5. **View Courses**: Lists all available courses with their schedules.

Code Breakdown :

1. Data Structures

The system uses two main structs:

```
typedef struct {
    int id;
    char name[50];
    int enrolledCourses[10];
    int courseCount;
} Student;


typedef struct {
    int id;
    char name[50];
    char schedule[20];
```

```
    int enrolledStudents[100];
    int studentCount;
} Course;
```

- **Student**:
  - id: Unique student identifier.
  - name: Student name.
  - enrolledCourses: List of enrolled course IDs.
  - courseCount: Tracks the number of enrolled courses.
- **Course**:
  - id: Unique course identifier.
  - name: Course name.
  - schedule: Course timing or schedule.
  - enrolledStudents: List of enrolled student IDs.
  - studentCount: Tracks the number of enrolled students.

---

2. User Menu

A **menu-driven interface** is implemented in the main function:

```
while (1) {
    printf("\n--- Course Registration System ---\n");
    printf("1. Add Student\n");
    printf("2. Add Course\n");
    printf("3. Enroll Student in a Course\n");
    printf("4. View Students\n");
    printf("5. View Courses\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1: addStudent(); break;
        case 2: addCourse(); break;
```

```
        case 3: enrollStudent(); break;
        case 4: viewStudents(); break;
        case 5: viewCourses(); break;
        case 6: exit(0);
        default: printf("Invalid choice! Try again.\n");
    }
}
```

This structure continuously displays options to the user until they exit.

---

3. Adding a Student

This function adds a student to the global students array. It prevents adding more students than the defined limit:

```
void addStudent() {
    if (studentCount >= MAX_STUDENTS) {
        printf("Maximum student limit reached!\n");
        return;
    }

    printf("Enter student ID: ");
    scanf("%d", &students[studentCount].id);
    printf("Enter student name: ");
    scanf(" %[^\n]s", students[studentCount].name);

    students[studentCount].courseCount = 0;
    studentCount++;

    printf("Student added successfully!\n");
}
```

---

4. Adding a Course

This function adds a course similarly to how students are added. It tracks the course ID, name, and schedule:

```
void addCourse() {
    if (courseCount >= MAX_COURSES) {
        printf("Maximum course limit reached!\n");
        return;
    }

    printf("Enter course ID: ");
    scanf("%d", &courses[courseCount].id);
    printf("Enter course name: ");
    scanf(" %[^\n]s", courses[courseCount].name);
    printf("Enter course schedule (e.g., Mon 10:00 AM): ");
    scanf(" %[^\n]s", courses[courseCount].schedule);

    courses[courseCount].studentCount = 0;
    courseCount++;

    printf("Course added successfully!\n");
}
```

---

5. Enrolling a Student

This is the core feature of the program, linking a student to a course by their IDs:

```
void enrollStudent() {
    int studentID, courseID, i, j;

    printf("Enter student ID: ");
    scanf("%d", &studentID);
    printf("Enter course ID: ");
    scanf("%d", &courseID);
```

```
    for (i = 0; i < studentCount; i++) {
        if (students[i].id == studentID) {
            for (j = 0; j < courseCount; j++) {
                if (courses[j].id == courseID) {
                    students[i].enrolledCourses[students[i].courseCount++] = courseID;
                    courses[j].enrolledStudents[courses[j].studentCount++] = studentID;

                    printf("Student enrolled in course successfully!\n");
                    return;
                }
            }
            printf("Course not found!\n");
            return;
        }
    }
    printf("Student not found!\n");
}
```

---

6. Viewing Data

To review the entered data, the program provides these functions:

- **View Students**: Displays all student IDs and names.
- **View Courses**: Displays course details, including schedules.

```
void viewStudents() {
    printf("\n--- Students ---\n");
    for (int i = 0; i < studentCount; i++) {
        printf("ID: %d, Name: %s\n", students[i].id, students[i].name);
    }
}

void viewCourses() {
    printf("\n--- Courses ---\n");
```

```
    for (int i = 0; i < courseCount; i++) {
        printf("ID: %d, Name: %s, Schedule: %s\n", courses[i].id, courses[i].name, courses[i].schedule);
    }
}
```

How to Use

1. Run the program.
2. Choose an option from the menu:
    - Add students or courses first.
    - Enroll students in the courses.
    - View the registered students and courses.
3. Exit the program when done.

Advantages

- Efficient use of **structs** for organizing data.
- Scalable, with adjustable limits for students and courses.
- Clear and modular functions for readability and reusability.

Limitations

- The system doesn't persist data (no database or file storage).

- Limited by predefined sizes (MAX_STUDENTS and MAX_COURSES).
- Lacks error handling for invalid inputs in nested loops.

Conclusion :

This simple course registration system demonstrates how you can manage relationships between entities in C using **structs** and arrays. You can enhance the program by adding persistent storage (files or databases) or by introducing dynamic memory allocation to remove size limitations.

Explore this example, and feel free to build upon it for more complex applications!

4o