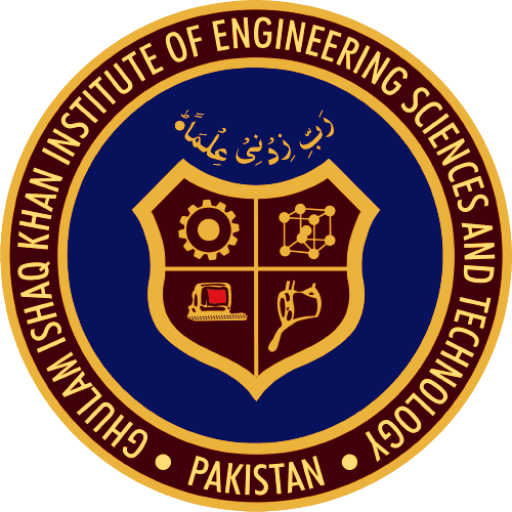
****

**CS112-Project**

**By: Abdul Rehman Asif:2024027**

**Umer Aslam:2024481**

**1.Introduction**

The C++ Course Management System is a console-based application designed to help manage academic courses efficiently. Leveraging core object-oriented programming concepts such as abstraction, inheritance, and polymorphism, the system enables users to add, update, delete, display, and search for courses. It utilizes structured data types, including enums, structs, and unions, to represent detailed course and semester information. Designed with modularity and maintainability in mind, the system provides a clear user interface and can be extended with features like file handling, dynamic memory, and sorting in future iterations.

**2. Key Components**

* **Data Structures:**
  + SemesterTerm (enum class): Defines the possible terms in a semester (FALL, SPRING, SUMMER, WINTER).
  + SemesterYear (enum class): Defines the possible years for a semester (Y2023, Y2024, Y2025, Y2026).
  + Semester (struct): Groups a SemesterTerm and a SemesterYear to represent a specific semester.
  + CourseInfo (union): A union that can hold either core course information (lectureHours) or elective course information (labHours). This is a memory-efficient way to store course-specific data, as a course is either core or elective, not both.
  + Course (struct): Represents a single course with attributes like name, code, credit hours, instructor, semester, core/elective status, and course-specific information using the CourseInfo union.
* **Classes:**
  + Manager (abstract base class): Defines the interface for course management operations using pure virtual functions. This enforces that any derived class *must* implement these functions. The presence of pure virtual functions makes Manager an abstract class, meaning you can't create objects of type Manager directly. It also includes a virtual destructor, which is good practice in inheritance hierarchies to ensure proper cleanup.
  + CourseManager (derived class): Inherits from the Manager class and provides concrete implementations for the course management operations. It uses an array (courses) to store Course objects and courseCount to track the number of courses.
* **Functions:**
  + getTermFromUser(), getYearFromUser(): Helper functions to get semester term and year input from the user, ensuring the input is converted to the appropriate enum type.
  + addCourse(): Adds a new course to the courses array, taking input from the user for all course details.
  + displayCourses(): Displays information for all courses in the courses array.
  + updateCourse(): Modifies the details of an existing course based on the course code.
  + removeCourse(): Removes a course from the courses array by shifting subsequent courses to fill the gap.
  + searchByCode(): Searches for a course by its code and displays its information.
  + listBySemester(): Lists courses offered in a specified semester.
  + main(): The entry point of the program. It creates a CourseManager object, presents a menu of options to the user, and calls the appropriate functions based on the user's choice.

**3. Code Walkthrough**

1. **Includes and Namespace:**
   * #include <iostream>: For input and output operations (e.g., cout, cin).
   * using namespace std;: To avoid repeatedly writing std:: before standard library elements.
2. **Data Structure Definitions:**
   * The code defines the enums (SemesterTerm, SemesterYear), structs (Semester, Course), and the union (CourseInfo) as described in the "Key Components" section. These structures organize the data related to courses and semesters.
3. **Class Definitions:**
   * **Manager Class:**
     + Declares pure virtual functions (addCourse(), displayCourses(), updateCourse(), removeCourse(), searchByCode(), listBySemester()). These functions define the operations that any course manager must support.
     + Has a virtual destructor (~Manager()). This is crucial for proper memory management when dealing with inheritance and pointers.
   * **CourseManager Class:**
     + Inherits from the Manager class.
     + Contains a private array courses[MAX\_COURSES] to store Course objects and an integer courseCount to keep track of the current number of courses.
     + Implements all the pure virtual functions declared in the Manager class, providing the actual logic for each course management operation.
     + Includes the helper functions getTermFromUser() and getYearFromUser() to handle user input for semester details.
4. **main() Function:**
   * Creates a CourseManager object dynamically using new and stores its address in a Manager pointer. This demonstrates polymorphism, as a Manager pointer can point to an object of its derived class (CourseManager).
   * Presents a menu-driven interface to the user using cout.
   * Uses a do-while loop to continuously display the menu and process user input until the user chooses to exit (option 7).
   * A switch statement handles the user's choice, calling the appropriate CourseManager member function.
   * Finally, it deletes the dynamically allocated CourseManager object using delete manager; to prevent memory leaks.

**4. Strengths**

* **Object-Oriented Design:** The code effectively uses OOP principles, making it modular, reusable, and easier to maintain.
* **Abstraction:** The Manager class provides an abstract interface, hiding the implementation details of course management.
* **Polymorphism:** The use of a Manager pointer to access a CourseManager object demonstrates polymorphism, allowing for potential future extensions with different types of managers.
* **Data Organization:** Structs and enums are used to organize data effectively, improving readability and maintainability.
* **Union for Memory Efficiency:** The CourseInfo union optimizes memory usage by storing either core or elective course information, but not both.
* **Clear User Interface:** The menu-driven interface is user-friendly, providing clear options for course management.
* **Error Handling:** The code includes basic error handling, such as checking for the maximum number of courses and handling invalid user input.

**5. Potential Improvements**

* **Dynamic Memory Allocation:** Instead of using a fixed-size array courses[MAX\_COURSES], consider using dynamic memory allocation (e.g., using std::vector) to handle a variable number of courses. This would make the system more flexible and prevent potential buffer overflows.
* **Input Validation:** Implement more robust input validation to handle various types of invalid input (e.g., non-numeric input when expecting numbers, invalid enum values).
* **File I/O:** Add functionality to save and load course data from a file, allowing data persistence.
* **Search and Sorting:** Implement more advanced search functionalities (e.g., search by instructor, search by course name) and sorting capabilities (e.g., sort courses by code, sort by name).
* **Exception Handling:** Use exception handling to manage errors more gracefully.
* **User Interface:** Consider using a more sophisticated user interface libraries
* **Modularity:** Further modularize the code by separating different functionalities into separate classes or files. For example, you could have a separate class for handling file I/O.
* **Templates:** Consider using templates to make the Manager class and potentially other parts of the code more generic, allowing them to work with different types of data if needed.

**6. Conclusion**

The provided C++ code is a good starting point for a Course Management System. It demonstrates fundamental OOP concepts and provides essential course management functionalities. By incorporating the suggested improvements, the code can be made more robust, flexible, and user-friendly.

**4.UML Diagram for code**

A diagram of a course

AI-generated content may be incorrect.