BS(AI) 2-A

OOPS SEMESTER PROJECT REPORT

Course Registration System Project Report

Submitted by:

- Muhammad Ibrahim
- Muhammad Yousaf

Enrollment Numbers:

- 01-136242-023
- 01-136242-055

Table Of Contents

Introduction

The **Course Management System** is a robust and interactive C++ application that simulates the administrative tasks typically associated with university course management.

The system supports student **registrations, manages courses and lecturers**, and enforces logical constraints like **enrollment limits and course activation conditions**.

It has been architected using modern object-oriented principles, structured exception handling, and STL-powered containers to ensure efficiency and maintainability.

This system has educational as well as practical value. For educators, it provides a complete example of OOP, design patterns, and file handling.

For developers, it lays a solid foundation that could be expanded into a GUI-based application or even a web-based system in future.

System Objectives

The primary goals of this system are:

- To enable students to register for academic courses within defined constraints.
- To allow lecturers to be associated with courses they instruct.
- To manage course validity based on enrollment thresholds.
- To persist data across sessions using file I/O.
- To handle runtime errors using structured exception handling.
- To demonstrate advanced object-oriented design, including polymorphism and design patterns.
- To use STL containers effectively for dynamic data management.

These objectives are carefully aligned with the broader principles of clean code, separation of concerns, and reusability.

System Overview

The system provides a menu-based interface for users to:

- Register a student into one of three courses: Programming, Databases, or Software Engineering.
- View detailed course information including participant lists and lecturer names.
- Display a list of courses that still have seats available.

- Display a summary of all enrolled courses for each student.
- Save and load all course and student data on program start and exit.
- Each course can have a maximum of 10 students, and courses require a minimum of 3 students to be considered valid.

These rules are hard-coded but clearly isolated in logic to allow easy modification if needed. The system is structured to simulate real-world scenarios in university administration while maintaining high software quality standards.

Design And Architecture

• Class Design

Class Name Responsibility Person Abstract base class defines a common interface (displayInfo()) for people. Student Extends Person.

Stores student-specific info like matriculation number and university affiliation.

Lecturer Extends Person. Stores academic title and contact information.

Course Manages a single course. Holds course name, a lecturer, and enrolled students. Course Catalog Singleton.

Manages the list of all available courses and their data.

Registration Exception Custom exception for handling invalid registrations.

File IO Exception Custom exception thrown on file I/O errors.

Each class encapsulates its responsibilities cleanly and exposes only necessary public interfaces.

Class constructors, member functions, and destructors are defined with attention to memory safety and object lifetime.

- ## Object Relationships and Polymorphism

The application uses a vector of base-class pointers (vector<Person*>) to store both Students and Lecturers.

This allows the use of polymorphic behavior when iterating over persons and calling displayInfo(), which is defined as a pure virtual method in the base class and overridden in derived classes.

This structure facilitates:

Dynamic type resolution at runtime clean separation of behavior while sharing a common interface Flexible future enhancements (e.g., adding more person types like Admin) The design adheres to the Leskov Substitution Principle, a core tenet of SOLID principles.

- ## Singleton Pattern for Course Management

The Course Catalog is implemented as a Singleton to ensure there is only one source of truth for managing course data throughout the system's lifecycle.
Key characteristics:

- Private constructors prevent direct instantiation
- Static getInstance() method provides access to the sole instance.
- Deleted copy constructor and assignment operator prevent copying.
- This ensures that all parts of the system share the same course registry, simplifying state management and eliminating bugs related to duplicated course data.

Features Implemented

✔ Student Registration: TU students may enroll in up to 3 courses, while external students are limited to 1.

✔ Lecturer Management: Lecturers are linked to specific courses with their names and academic titles displayed.

✔ Course Enrollment Cap: Each course can accept a maximum of 10 students. Registrations beyond this are rejected.

✔ Course Validity Check: Courses must have at least 3 students to be considered active or scheduled.

✔ Dynamic Listing: The system dynamically lists fully booked or under booked courses.

✔ Persistent Storage: All data is saved on exit and loaded on program startup.

✔ Polymorphic Output: All Person-derived classes implement displayInfo() to enable polymorphic output.

✔ User-friendly Interface: Simple and clear menu interface with error handling and input validation.

✔ Operator Overloading: Custom << operator for clean printing of Student and Lecturer data.

✔ Exception Handling: Critical runtime errors are managed through structured and meaningful exceptions.

File I/O and Data Persistence

Persistence is implemented using plain-text files: students.txt: Stores each student's name, university, matriculation number, and enrolled courses.

courses.txt: Stores course name, lecturer info, and list of participants.

Data is loaded into memory on startup and saved on every change.

This makes the application stateful and user-friendly.

The system handles errors like: Missing files Corrupt data entries Write failures due to permission errors Custom exceptions (FileIOException) provide clear feedback when such issues arise, instead of allowing silent data corruption.

Exception Handling Mechanism

Exception handling in this system is elegant and structured.

Two custom exceptions are defined: Registration Exception:

Raised when a registration violates rules (e.g., over-capacity, wrong university)

FileIOException: Raised when the file system fails during load/save operations.

The system uses try-catch blocks to: Catch and report errors cleanly to the user

Prevent crashes or undefined behavior Ensure stability and reliability

Operator Overloading for Stream Output

Custom operator<< overloads are implemented for:

Student Lecturer This allows objects to be directly streamed into output like: cpp Copy Edit cout << student;

The output is clean, readable, and formatted for presentation, making debugging and logging simpler.

Example output: yaml Copy Edit Student: John Doe (Email: john@example.com) Matriculation No.: 12345 | University: TU Enrolled in: Programming, Databases

Testing and Validation Strategy

A comprehensive set of tests was run to ensure system robustness:

Functional Testing

- ✔ Register TU student for 3 courses → Success
- ✔ Register TU student for 4th course → Throws exception
- ✔ Register external student for 2 courses → Throws exception
- ✔ Register students beyond course capacity → Displays full
- ✔ Load/save operations → Data persists as expected

Error Handling Tests

- ✔ Invalid input types (e.g., letters in place of numbers) → Input rejected gracefully
- ✔ Corrupt files → Program catches errors and informs user Performance and Memory Tests
- ✔ Memory usage with large number of students and courses (stress test) remained within bounds
- ✔ Destructor behavior validated via debug logs (for dynamic arrays and STL)

Conclusion

This Course Management System is a complete and extensible application showcasing best practices in C++ development.

It successfully implements: Advanced OOP concepts Polymorphism STL container usage File I/O and persistence Exception handling Design patterns (Singleton) Operator overloading

The architecture is modular, making it easy to test, maintain, and extend.

With the addition of a GUI or web interface, it can be scaled into a real-world educational tool.

As it stands, it serves as an excellent academic and professional example of clean, object-oriented software engineering in C++