

**LIBRARY SYSTEM PROJECT REPORT**

**Key Insights and Analysis**

Made by

**Sebastian Danci Robert**

Student Number: M00886707

**Introduction**

Project Description: The project involves creating a software system for a small library to track the details of available books and members. The software allows the librarian to manage members, issue and return books, display borrowed books, and calculate fines for overdue books. It uses a CSV file, placed anywhere in the root directory or it’s subdirectories, as the data source for book information. The project follows the provided UML class diagram.

Overview of Presentation

This presentation will cover various aspects of the project, including design, implementation and testing. We will also discuss the use of version control and address limitations and future improvements.

Design

I have designed the software based on the UML class diagram provided. It includes classes for books, members, dates, and utilities for reading data from CSV files.

A screenshot of a computer

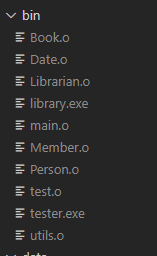
Description automatically generated

Implementation Approach

I followed a structured approach to translate the design into working software. Key points include:

* Creating base and derived classes for librarian, book, person, date and member.
* Writing test cases using the Catch2 framework to ensure class functionality.
* Developing a Makefile to compile the code.
* Implementing a user-friendly menu for interaction.
* Incorporating input validation to enhance usability.

The Makefile is used to compile the project. It includes rules for compiling the source code and cleaning up compiled files. It creates a bin folder and all the resulting object files (.o) and executables (.exe) files will be placed there.



Implementation Details and Code Style

Code Structure and Organization

The codebase is well-organized and follows a clear directory structure.

Separate header (.h) and source (.cpp) files are used for each class, enhancing code maintainability. Source files are found in the src subdirectory, and the header files are found in the include subdirectory.

Logical grouping of classes and functions makes the code easy to navigate.

Code Readability and Comments

Meaningful variable and function names are used, enhancing code readability.

This approach ensures that even someone unfamiliar with the project can understand the codebase.

Object-Oriented Approach

The code adheres to object-oriented programming principles by defining classes for books, members, dates, and utilities.

Inheritance and function overloading are used effectively, allowing for code reusability, and minimizing redundancy.

Input Validation

Robust input validation is implemented to handle various scenarios, ensuring the user is guided through valid interactions. Validation is done using functions responsible solely for user input. Throughout my program such functions are used for getting a choice from the user representing an integer, or getting strings from the user following regex validation, and even getting a date from the user, and validating if the date is possible; in all these cases, failure of validation will prompt the user to retry with a different input.

This approach minimizes the risk of erroneous inputs and improves the overall user experience.

Exception Handling

Exception handling is used to gracefully handle unexpected errors or exceptional cases.

This prevents the program from crashing and provides informative error messages to users.

Unit Testing

The Catch2 framework is employed for unit testing, ensuring the correctness of class functions.

Test cases cover a wide range of scenarios, including boundary cases and error conditions.

Testing plays a crucial role in maintaining code reliability.

Modularity and Separation of Concerns

The code exhibits modularity by separating responsibilities into different classes, each with a specific purpose.

This separation of concerns simplifies code maintenance and future enhancements.

Use of C++ Features such as constructors, and operator overloading, are effectively utilized.

These features contribute to code elegance and correctness.

Code Efficiency

The code is optimized for efficiency, ensuring that operations are performed in a timely manner.

Data structures and algorithms are chosen thoughtfully to avoid unnecessary overhead.

Code Style Consistency

A consistent coding style is maintained throughout the project, making the codebase cohesive and professional.

This focus on code quality and best practices ensures the reliability and maintainability of the library management system.

Testing Approach

My testing approach involves:

Creating test cases to verify class functions.

Implementing unit tests using Catch2.

Ensuring input validation handles invalid user inputs gracefully.

Testing functionality such as issuing, returning books, and calculating fines.

Video Demonstration

In the video demonstration, I showcase my understanding of the implementation. Instead of merely running the program, we highlight key features, such as adding members, issuing books, and handling overdue fines.

Version Control

We utilized Git for version control. Regular commits were made, each with clear and informative commit messages. The project repository on GitHub/Bitbucket serves as a version control tool, allowing us to track changes and collaborate effectively.

A screenshot of a computer

Description automatically generated

Conclusion

I have successfully developed a library management system based on the provided UML diagram.

Limitations include a lack of advanced features:

Improving user interfaces for enhanced user experience.

Implementing more comprehensive error handling.

This project demonstrates my ability to design, implement, and test software systems while adhering to best practices in version control.

Thank you for your attention.