Luna's Library System

Documentation and Code Overview

Angel E Luna

June 12, 2025

Contents

1 Introduction

Luna's Library System is a console-based application developed in C++ that simulates the core functionalities of a digital library. The system allows users to register, log in, borrow and return various types of library items such as books, theses, and magazines, and to view their borrowing history. All user and library data are stored persistently using CSV files, ensuring that information is retained between sessions.

The project was designed to provide a practical demonstration of object-oriented programming principles, file handling, and menu-driven user interfaces in C++. By tackling common challenges found in real-world library management—such as user authentication, data consistency, and resource tracking—this system serves as both a learning tool and a foundation for further development. The modular structure of the code makes it easy to extend with new features or adapt to different requirements.

This documentation provides a comprehensive overview of the system's design, functionality, and usage, along with guidelines for future development and extension.

2 Project Motivation and Rationale

The decision to develop Luna's Library System was driven by several key factors:

- **Practical Application of C++ Concepts:** This project provides a comprehensive exercise in object-oriented programming, file I/O, and data structures, all of which are fundamental to mastering C++. By implementing a real-world system, the project bridges the gap between theoretical knowledge and practical skills.
- **Relevance to Everyday Needs:** Library management is a classic problem that remains relevant in both educational and professional contexts. Designing a digital library system simulates challenges faced in real information systems, such as user authentication, data persistence, and resource management.
- **Scalability and Extensibility:** The project was chosen for its potential to be extended with new features, such as additional file types, advanced search, or even a graphical interface. This makes it an ideal foundation for future learning or collaborative development.

- **Personal Interest and Challenge:** Building a library system is both intellectually stimulating and rewarding. It requires careful planning, attention to detail, and creative problem-solving, making it an ideal project for honing software development skills.

3 Project Structure

- `main.cpp` – Main program logic and menu handling
- `bibliofiles.hpp` – Base and derived classes for library items (Book, Thesis, Magazine)
- `user.csv` – Stores user data
- `books.csv`, `thesis.csv`, `mags.csv` – Store library items

4 Main Features

- User registration and login
- Borrowing and returning files
- Viewing file information and fragments
- Persistent storage using CSV files
- User history tracking

5 Class Overview

5.1 BiblioFiles and Derived Classes

```
class BiblioFiles {
protected:
    std::string idfile, title, author, filetype, fragment;
    int publicationyear;
    bool availability;
public:
    // Methods for getting info, showing fragments, etc.
};
```

Listing 1: BiblioFiles Base Class

Book, **Thesis**, and **Magazine** inherit from `BiblioFiles` and add specific fields and methods.

5.2 User Class

```
class User {
protected:
    std::string history;
    std::string name, password;
    std::string borrowedfiles;
public:
    void borrowfile(BiblioFiles* file);
    void returnfile(BiblioFiles* file);
    // Other user-related methods
};
```

Listing 2: User Class

6 Program Flow

6.1 Startup

1. Loads user data from `user.csv`.
2. Displays the main menu: Login, Register, Exit.

6.2 Login and Registration

- On login, verifies username and password.
- On registration, checks for unique username and appends new user to `user.csv`.

6.3 Library Data Loading

After successful login, the program loads library data from `books.csv`, `thesis.csv`, and `mags.csv`.

6.4 Main Application Loop

1. Shows user menu: View files, Search, Borrow, Return, History, User Settings, Exit.
2. Handles user actions with nested menus and input validation.
3. Updates user and library data in memory and in CSV files.

7 File Operations

- Reading: Uses `std::ifstream` and `std::getline` to load data.
- Writing: Uses `std::ofstream` (with `std::ios::app` for appending or `std::ios::trunc` for overwriting).
- Updates: Reads all lines, modifies in memory, then rewrites the file.

8 Example: Borrowing a File

```
void User::borrowfile(BiblioFiles* file) {
    if (!history.empty()) history += ";";
    history += file->getidfile();
    // Update user.csv with new borrowed file and history
}
```

9 Error Handling

- The program checks for file open errors and invalid data.
- On fatal errors (e.g., missing CSV files), the program prints an error and exits.
- Input is validated at each menu to prevent invalid actions.

10 Data Consistency

The system reads the entire CSV file into memory on startup and writes back any changes immediately. This ensures that all user and library data is consistent and up-to-date.

11 Object-Oriented Programming Concepts Used

11.1 Inheritance

Inheritance allows a class to use the properties and methods of another class. This promotes code reuse and logical hierarchy. A child class inherits from a parent class and can also add its own functionality.

11.2 Access Modifiers

Access modifiers control the visibility of class members:

- **public**: Accessible from anywhere.
- **private**: Accessible only within the same class.
- **protected**: Accessible within the class and its subclasses.

They help with encapsulation and data protection.

11.3 Method Overloading and Overriding

Overloading: When a class has multiple methods with the same name but different parameter lists. This is resolved at compile time.

Overriding: When a subclass redefines a method from its parent class to provide specific behavior. This is resolved at runtime, usually using `virtual` in languages like C++.

11.4 Polymorphism

Polymorphism allows the same interface to be used for different data types or behaviors. It enables flexibility and code reuse. There are two main types:

- **Compile-time polymorphism** (e.g., method overloading)
- **Runtime polymorphism** (e.g., method overriding using base class pointers or references)

11.5 Abstract Classes

An abstract class cannot be instantiated on its own. It often contains one or more pure virtual functions, which means any subclass must implement those functions. It serves as a blueprint for other classes.

11.6 Operator Overloading

Operator overloading lets you redefine the behavior of standard operators (like `+`, `-`, `==`) when they are used with objects of your own classes. This makes your custom types easier to use and understand.

12 Future Work and Enhancements

Potential future enhancements include:

- Adding more file types (e.g., audiobooks, videos)
- Implementing advanced search and filtering options
- Developing a graphical user interface (GUI)
- Adding networked multi-user support
- Implementing a web-based interface

13 Conclusion

Luna's Library System is a robust and flexible foundation for understanding and developing library management systems. It covers essential programming concepts and provides a platform for future enhancements and learning.

14 Program Execution Examples

This section provides screenshots and sample outputs to illustrate how Luna's Library System appears during execution.

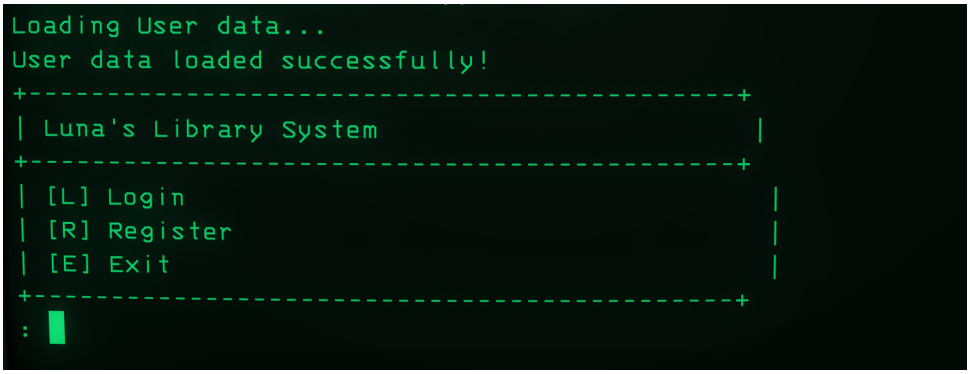

Figure 1: Screenshot 2025-06-12 at 20.52.09.png — Main menu displayed at program startup.

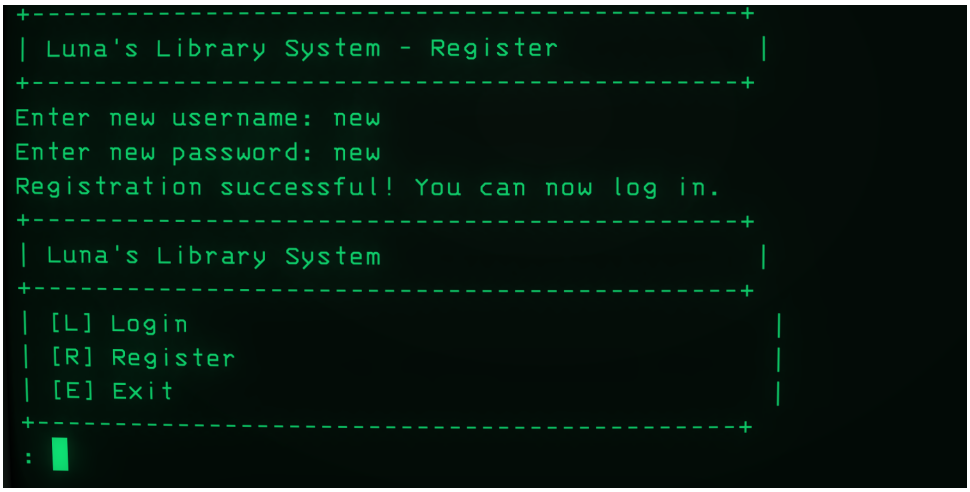

Figure 2: Screenshot 2025-06-12 at 20.52.34.png — User registration process.

14.1 Startup and Main Menu

14.2 User Registration

14.3 Login

14.4 Viewing Library Files

14.5 Borrowing a File

14.6 Returning a File

14.7 User History

14.8 User Settings

14.9 Exiting the Program

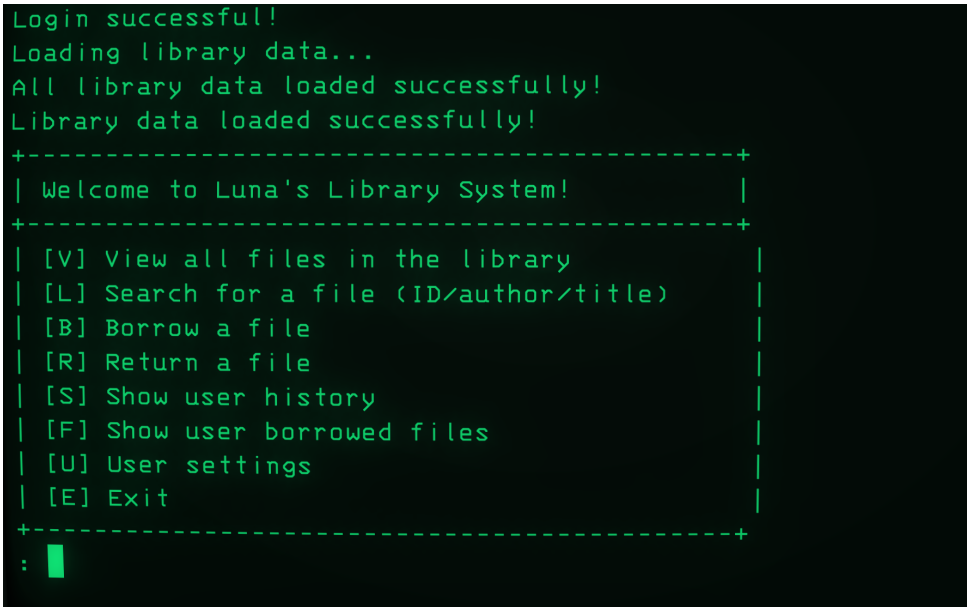

Figure 3: Screenshot 2025-06-12 at 20.53.00.png — User login prompt.

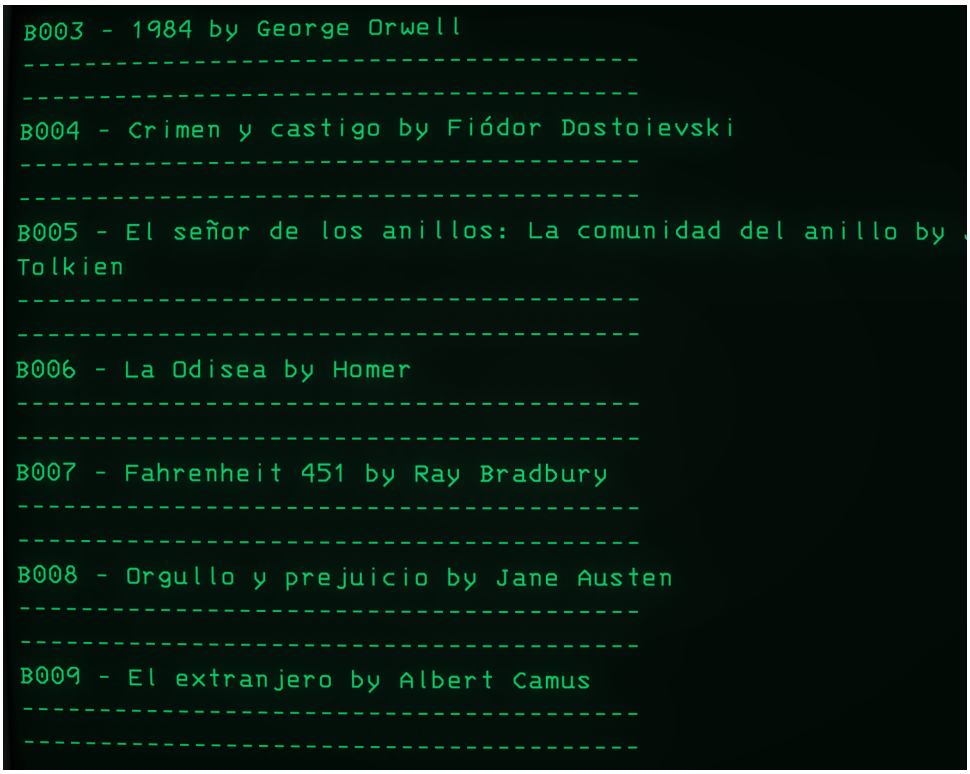

Figure 4: Screenshot 2025-06-12 at 20.53.23.png — Viewing all files in the library.

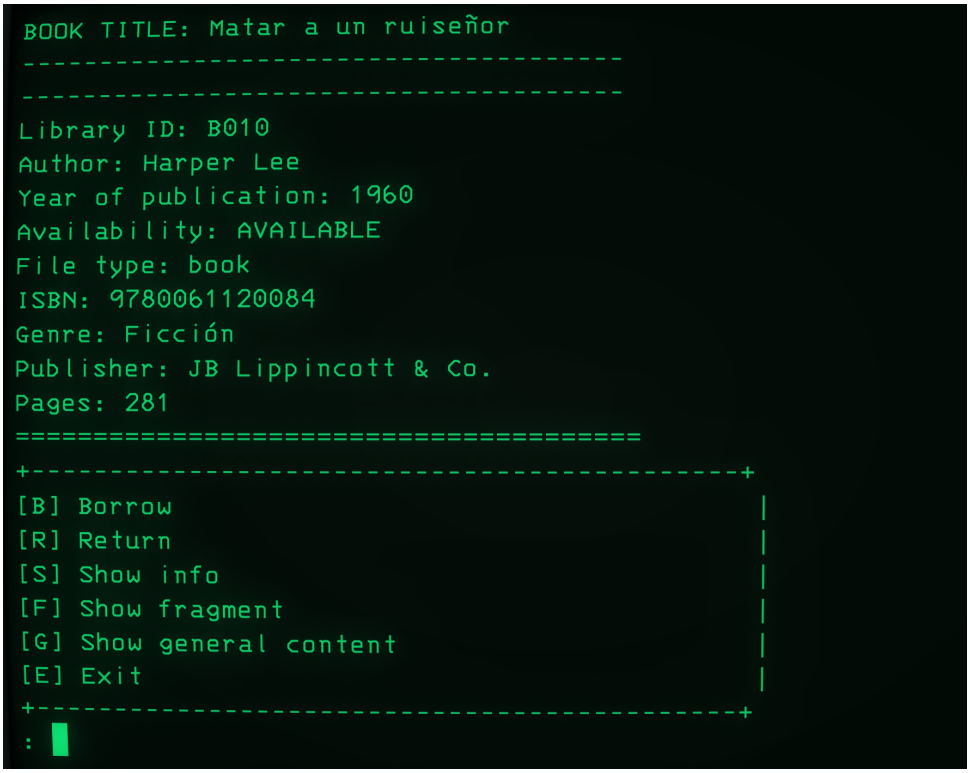

Figure 5: Screenshot 2025-06-12 at 20.53.52.png — Borrowing a file from the library.

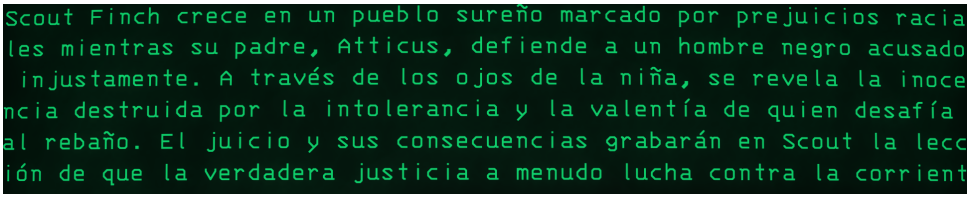

Figure 6: Screenshot 2025-06-12 at 20.54.10.png — Returning a borrowed file.

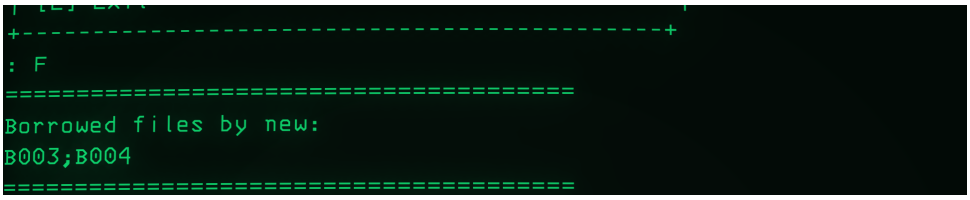

Figure 7: Screenshot 2025-06-12 at 20.55.30.png — Viewing user borrowing history.

Figure 8: Screenshot 2025-06-12 at 20.55.55.png — User settings menu.

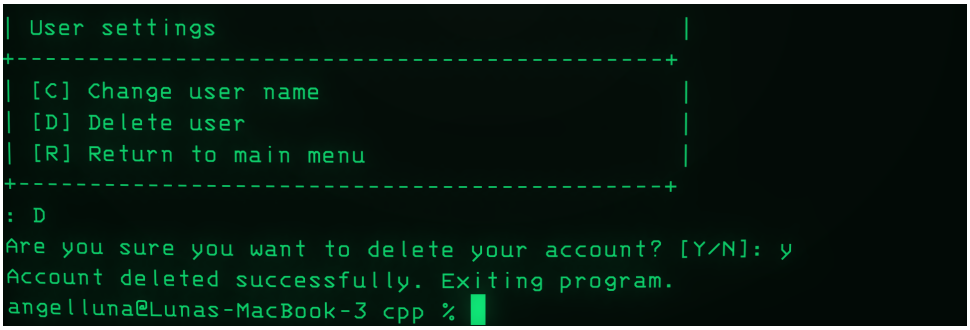

Figure 9: Screenshot 2025-06-12 at 20.56.20.png — Exiting the program.

References

[1] 2024. C++ reference. Accessed: 2024-06-12. https://en.cppreference.com/w/.

[2] 2024. Cplusplus.com. Accessed: 2024-06-12. https://www.cplusplus.com/.

[3] 2024. The LaTeX project. Accessed: 2024-06-12. https://www.latex-project.org/.