# Project Report: Library Management System

Submitted By: Syeda Fatima Waseem 24K-0924, Syed Ikrash Ahmed 24K-0998
Course: OOP
Instructor: Shafique Rehman
Submission Date: May 11, 2025

## 1. Executive Summary

**- Project Overview:** This project aims to develop a comprehensive Library Management System (LMS) that allows users to borrow, return, and renew books, while also providing librarians with tools to manage books and user accounts. The system incorporates object-oriented programming principles in C++ to model users (Premium, Normal, and Librarian), books, and their interactions. The system also includes a login mechanism with password encryption for secure access.

## 2. Introduction

**- Background:** Traditional library systems often lack efficiency in managing user accounts, tracking borrowed books, and calculating fines. This project introduces a digital solution to automate these tasks, ensuring accuracy and ease of use. The system supports multiple user types, each with distinct privileges and limitations, and includes features like fine calculation, book renewal, and account management.

**- Objectives of the Project:**

- Develop a user-friendly interface for library operations.
- Implement secure login functionality with password encryption.
- Enable users to borrow, return, and renew books.
- Allow librarians to manage books and user accounts.
- Calculate fines for overdue books based on user type.

# 3. System Description

**- Original System Rules:**

- Users can borrow books for a fixed period (14 days).
- Fines are calculated based on overdue days and user type.
- Librarians can add or remove books and user accounts.

**- Innovations and Modifications:**

- Introduced user types (Premium, Normal, Librarian) with varying privileges.
- Implemented password encryption for secure login.
- Added fine calculation logic tailored to user types.
- Enabled book renewal with limits based on user type.

# 4. AI Approach and Methodology

**- AI Techniques Used:** While the project does not heavily rely on AI, it uses algorithmic logic for fine calculation, book renewal limits, and password encryption. The system evaluates user actions (e.g., borrowing, returning) and updates the database accordingly.

**- Algorithm and Heuristic Design:**

- Fine Calculation: Uses date comparison to determine overdue days and applies fines based on user type (Premium: Rs 5 per 15 days, Normal: Rs 0.5 per day).
- Password Encryption: Implements a simple Caesar cipher for encoding passwords.
- Book Renewal: Limits renewals (Premium: 3 times, Normal: 1 time).

**- Performance Evaluation:** The system efficiently handles user interactions, with fine calculations and book management operations.

## 5. System Mechanics and Rules

**- Modified System Rules:**

- Premium Users: Can borrow up to 10 books, renew 3 times, and pay Rs 5 per 15 days overdue.
- Normal Users: Can borrow up to 3 books, renew once, and pay Rs 0.5 per day overdue.
- Librarians: Can manage books and user accounts but cannot borrow books.

**- Turn-based Mechanics:** Users interact with the system via a menu-driven interface, selecting options to perform actions like borrowing, returning, or searching for books.

## 6. Implementation and Development

**- Development Process:** The system was developed iteratively, starting with core classes (User, Book, System) and gradually adding features like login security and fine calculation. Testing was performed at each stage to ensure functionality.

**- Programming Languages and Tools:**

- Programming Language: C++
- Libraries: <iostream>, <fstream>, <vector>, <ctime>
- Tools: GitHub and Git for version control

**- Challenges Encountered:**

- Managing file I/O for user and book data.
- Implementing secure password handling.
- Ensuring fine calculation accuracy across different date scenarios.

# 7. Team Contributions

**- Team Members and Responsibilities:**

- Fatima:
    - Login system setup
        - login.hpp header file
        - signUp() function in system.hpp
    - book class (in allClasses.hpp)
    - user class & its child classes (allClasses.hpp)
    - functions in system.hpp:
        - searchBookName(), searchAuthorName() & searchBookId()
        - displayAllBooks(), displayBorrowedBooks() & displayAvailableBooks()
        - borrowBook(string idOfUser), returnBook(string idOfUser) & renewBook(string idOfUser)
- Ikrash:
    - main.cpp
    - user and librarian menu (inside system.hpp)
    - handled filing I/O functions in system.hpp
        - loadBooks()
        - saveBooks()
        - loadPremiumUsers()
        - loadNormalUsers()
        - loadLibrarian()
        - saveUsers()
        - deleteYourAccount()

# 8. Results and Discussion

**- System Performance:** The system successfully handles all specified operations, with fine calculations and book renewals working as intended. The login system ensures secure access, and the menu-driven interface is intuitive for users.

**- Limitations and Future Work:**

- Add graphical user interface (GUI) for better usability.
- Extend password encryption to more secure methods.
- Implement multi-threading for concurrent user access.

# 9. OOP Principles Used

This project extensively utilizes Object-Oriented Programming (OOP) principles to model real-world entities (users, books, and the library system) and their interactions. Below are the key OOP concepts applied:

**1. Encapsulation**

- Definition: Bundling data (attributes) and methods (functions) that operate on the data into a single unit (class), while restricting direct access to some components.
- Implementation:
    - Classes (User, Book, System, LoginSystem) encapsulate their respective data and behaviors.
    - Example: The Book class encapsulates attributes like bookID, title, author, and methods like borrowBook(), returnBook().
    - The LoginSystem class hides password-handling logic (encode(), getPassword()) from external access.
    - Access Specifiers (private, protected, public) ensure data integrity by controlling access.
    - Example: borrowedBooks in PremiumUser is protected to prevent unauthorized modifications.

**2. Inheritance**

- Definition: Creating a hierarchy where derived classes inherit properties and behaviors from a base class.
- Implementation:
    - Base Class (User) defines common attributes (userID, name, contactNum) and abstract methods (borrowBook(), returnBook()).

- Derived Classes (PremiumUser, NormalUser, Librarian) extend User with specialized behaviors:

- PremiumUser and NormalUser override methods like calculateFine() with type-specific logic.

- Librarian restricts book-related actions (e.g., borrowBook() prints "not a valid librarian function").

**3. Polymorphism**

- Definition: Allowing objects of different classes to be treated as objects of a common superclass, with methods behaving differently based on the actual object type.
- Implementation:
  - Virtual Functions: The User class declares virtual methods (e.g., payFine(), displayBooksBorrowed()), overridden in derived classes.

  - Example: PremiumUser::calculateFine() computes fines per 15 days, while NormalUser::calculateFine() uses per-day rates.

  - Abstract Class (User): Contains pure virtual functions (= 0), enforcing implementation in derived classes.

**4. Abstraction**

- Definition: Hiding complex implementation details and exposing only essential features.
- Implementation:
  - Abstract Methods: User class abstracts actions like borrowBook() without implementation.

  - Simplified Interfaces: The System class provides high-level methods (e.g., userMenu(), LibrarianMenu()) that internally handle low-level operations (file I/O, fine calculations).

**5. Association & Composition**

- Definition: Modeling relationships between classes (e.g., "has-a" relationships).

- Implementation:
    - Composition: The System class "has" Book and User objects (stored in vectors allBooks, allUsers).

    - Aggregation: System uses LoginSystem to manage authentication without owning it.

**6. Operator Overloading**

- Definition: Defining custom behavior for operators (e.g., << for output).
- Implementation:
    - operator<< overloaded for Book and NormalUser to customize display formats.

Conclusion

The project adheres to OOP principles to create a modular, maintainable, and scalable system. Encapsulation ensures data security, inheritance promotes code reuse, polymorphism enables flexible behaviors, and abstraction simplifies complex operations. These principles collectively enhance the system's robustness and extensibility for future enhancements.

# 10. References


- C++ Documentation.
- Online resources for file I/O and date handling in C++.
- Password encryption techniques.