Coursework 1:

Library Management System

Nouar Mahamoud

M00873504

In this coursework, we delve into the design and implementation of a Library Management System. The coursework explores creating software to library tasks. From class design to functionalities like member registration and book management, etc. The coursework also emphasizes file handling, error detection, and collaborative development using version control systems.

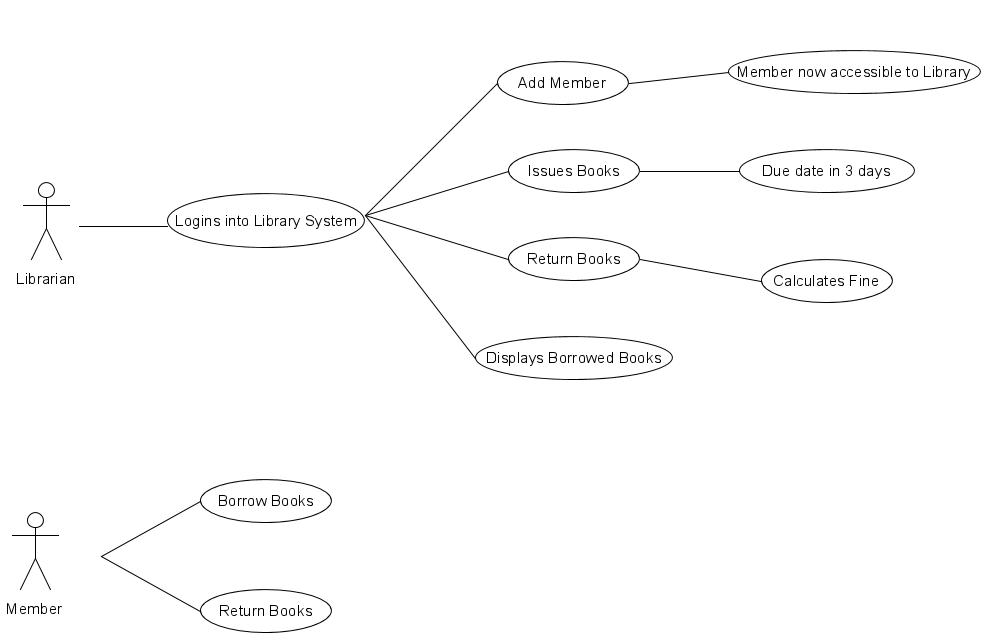


Figure 1

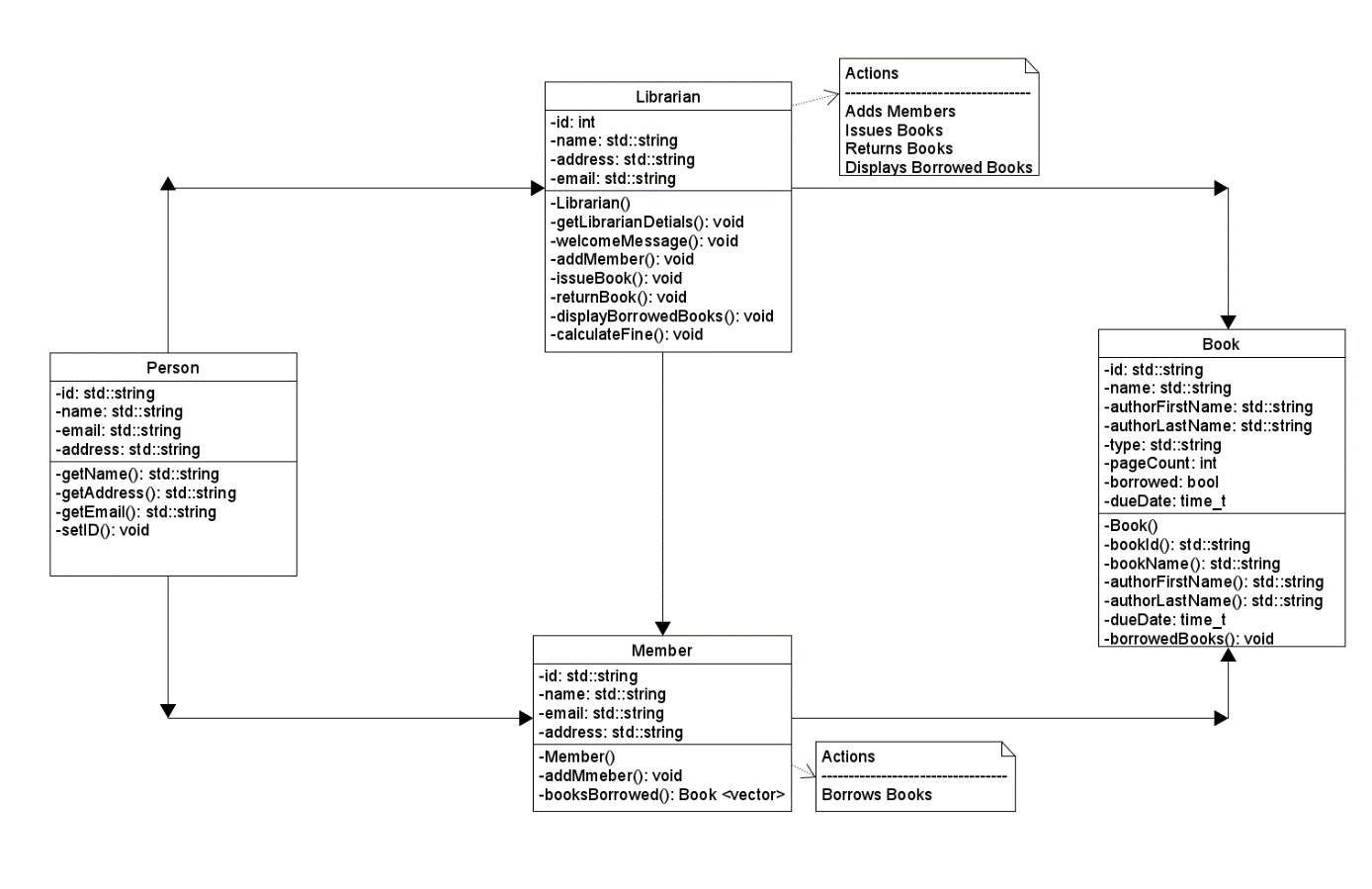


Figure 2

In Figure 1, the UML diagram is essential for the understanding of logic and steps towards the actions the Librarian (User) chooses. Using the diagram has meant that I could implement concepts into the code to accurately reflect the Library Management System. It incorporates:

1. Class Representation:
   * Identifies how each class in the UML diagram (Librarian, Book, Member, etc) is translated into the corresponding C++ class.
   * Examines how class attributes (e.g., name, ID, etc) are defined as variables.
   * Class operations are implemented as functions.
2. Relationship Mapping:
   * Relationships classes in the UML diagram are represented by using data structures and constructs.
3. Functionality Implementation:
   * Functionalities of the library system, as depicted in the UML diagram, are implemented.
   * Functions like adding members, issuing books, returning books, displaying borrowed books, and any other actions are outlined in the diagram.
   * The flow of logic and data within these functions, referencing the UML diagram are to clarify design decisions.

Alike Figure 1, Figure 2 has the transformed design into a functional library management system. The code translation focusing on the key elements and relationships utilises the classes to mirror the UML model's structure and behaviour.

1. Translating the Concepts:

* Librarian: Functions like Librarian(), getLibrarianDetails() and welcomeMessage(), correspond directly to the diagram's operations, allowing for librarian information and greetings.
* Book: Variables like id, name and authorFirstName and borrowed mirror the diagram's attributes, while functions like Book() and bookName() provide access to crucial book details.
* Member: Functions like Member(), addMember() and borrowedBooks() align with the diagram's operations, enabling member registration, book borrowing, and tracking.

2. Relationships Take Shape:

* Associations: The connections between classes, like the one between Member and Book through borrowsBooks(), are implemented using vectors. This reflects the UML's dynamic system that can handle multiple members and borrowed books with ease.

3. Functionalities in Action:

* Adding Members: I’ve Implemented user input validation and ID generation, as required, as this ensures accurate member registration and avoids duplicate IDs.
* Issuing and Returning Books: The issueBooks() and returnBooks() functions involve checks for book availability, member validity, and potential overdue fines. These functions, along with updates to the borrowed status, reflect the core functionalities of the library management system.
* Other Features: Additional features such as displaying borrowed books or calculating fines. These features would further showcase the system's capabilities.

Makefile:

Makefiles are essential for automating building processes in software development. They save time, enforce consistency, track dependencies, and provide flexibility for diverse project needs. In the image below is a makefile that compiles various C++ files to have an output compile. This simplifies the build process by specifying dependencies and commands needed for each step.

