



华南理工大学
South China University of Technology

Data Structure Project Report

Topic: Book Library Management System

Faculty: School of International Education

Program: Course Work of Data Structures

Names: Sudipta Sotra Dhar, Subodh Pokhrel, Ibrahim Alfaqih, Walid KH.J Suleiman

Student ID's: 202169990264, 202169990044, 202169990079, 202169990015

Lecturer: Professor Patrick Chan

Course Code: CST041C

Credit: 1.0

Start/End Date: March 20th, 2023 - June 20th, 2023

<p>Teacher's Comment</p>	<p>Signature:</p> <p>Date:</p>
<p>Score</p>	
<p>Remark</p>	

TABLE OF CONTENTS

S.N	Contents	Page No:
1	Background	3
2	Design Principle	3-14
3	Model Discussion	14-22
4	Demonstration	22-30
5	Conclusion	30
6	References	30
7	Appendix	31-40

Library Management System

Background:

The Library Management System project serves as a prime example of the practical implementation of data structures and algorithms in a real-world scenario. By efficiently organizing books, borrowers, and users, this system demonstrates the power of utilizing appropriate data structures and employing algorithms for various operations.

Design Principals:

1. Book Class

The Book class represents a book in the library system. It has protected member variables such as title, author, ISBN, type, `available_copies`, and `reserved_id`. The protected keyword indicates that these variables can be accessed by derived classes. The class has a constructor that takes the book's title, author, ISBN, type, and `available_copies` as parameters. When a new Book object is created, the constructor initializes these member variables with the provided values. The constructor uses an initialization list to set the member variables directly, which is a more efficient way of initializing variables compared to setting them inside the constructor body. The title, author, ISBN, type, and `available_copies` member variables are assigned the corresponding values passed as arguments to the constructor. The `reserved_id` vector is not initialized explicitly in the constructor, so it will be empty by default. Overall, this constructor allows convenient creation of Book objects by providing the necessary information during object initialization.

1.1 Display methods:

The `display()` function in the Book class is responsible for printing the details of a book to the console. It outputs information such as the book's title, author, ISBN, type, and the number of available copies. Additionally, it displays the reserved IDs associated with the book. First, the function prints the title, author, ISBN, and type using the corresponding member variables. Then, it checks if there are any reserved IDs stored in the `reserved_id` vector. If the vector is empty, it displays the message "No reservation". If there are reserved IDs, it iterates through the vector using a loop and prints each ID number to the console.

This function provides a convenient way to view the details of a book, including its availability and reservation status. It assists users and administrators in accessing essential information about the book in the library system.

1.2 Getter methods:

set of getter methods within the Book class. These methods allow external access to the private member variables of the class, providing a way to retrieve their values. The `getTitle()` method returns the title of the book as a string, `getAuthor()` returns the author's name, `getISBN()` returns the ISBN number, `getType()` returns the book's type, `getAvailableCopies()` returns the number of available copies as an integer, and `getReservedIds()` returns a vector of integers representing the reserved IDs associated with the book. These getter methods play a crucial role in encapsulation by providing controlled access to the internal state of a Book object, enabling other parts of the program to retrieve specific information about a book, such as its title, author, ISBN, type, available copies, and reserved IDs.

1.3 Setter methods:

a set of setter methods within the Book class. These methods allow external manipulation of the private member variables of the class, enabling the modification and update of their values. The `setTitle()` method sets the title of the book based on the provided string parameter, `setAuthor()` sets the author's name, `setISBN()` sets the ISBN number, `setType()` sets the book's type, and `setAvailableCopies()` sets the number of available copies based on the provided integer parameter. Additionally, the `setReservedIds()` method sets the reserved IDs associated with the book using a vector of integers provided as a parameter. The `addReservedId()` method allows adding a new reserved ID to the existing collection. These setter methods provide a way to modify the internal state of a Book object, allowing for flexibility and dynamic updates to the book's attributes as needed within the application.

2. User Class:

User class with private member variables `id`, `name`, `type`, and `password`. These variables store information related to a user's identification, name, user type, and password, respectively. The class also includes a constructor that initializes these member variables based on the provided arguments: `id`, `name`, `type`, and `password`. The constructor takes an `id` of type `int`, a `name` of type `string`, a `type` of type `string`, and a `password` of type `string`. When an instance of the User class is created, these values are used to set the corresponding member variables. The private access specifier ensures that these member variables cannot be directly accessed or modified from outside the class. To access or modify these variables, getter and setter methods would need to be defined within the class. Overall, the User class encapsulates user-related data and provides a way to create user objects with their associated attributes.

2.1 Getter methods:

getter methods for the User class includes methods to allow access to the private member variables of the class from outside the class itself. The `getId()` method returns the `id` of the user, which is of type `int`. The `getName()` method returns the name of the user, which is of type `string`. The `getType()` method returns the type of the user, also of type `string`. Finally, the `getPassword()` method returns the password of the user, again of type `string`. By using these getter methods, other parts of the program can retrieve the values of these member variables for a particular User object. The `const` keyword after the function declaration indicates that these methods do not modify the state of the User object and are therefore considered as `const` member functions. Overall, the getter methods provide a way to access the private data stored within the User class in a controlled manner, without directly exposing the member variables to external code.

2.2 Setter methods:

The setter methods allow the modification of the private member variables of the class. The `setId()` method sets the `id` of the user to the provided integer value. The `setName()` method sets the name of the user to the provided string value. The `setType()` method sets the type of the user to the provided string value. Lastly, the `setPassword()` method sets the password of the user to the provided string value. The `display()` method is responsible for outputting the details of a User object. It prints the `id`, `name`, and `type` of the user to the console. The `const` qualifier ensures that the method does not modify the state of the object. Together, the setter methods allow the modification of user information, while the `display` method provides a way to view the details of a User object. These methods contribute to the functionality of the User class by enabling data manipulation and information retrieval.

3. Borrower Class:

The Borrower class has a private data member `id` of type `int`, which represents the unique identifier of the borrower. It also has a private data member `type` of type `string`, which specifies the type or category of the borrower. Additionally, there is a private data member `borrowedBooks` of type `vector<string>`, which stores the titles of the books borrowed by the borrower. The constructor `Borrower(const int& id, const string& type)` initializes the `id` and `type` of the borrower with the values. The Borrower class encapsulates the information about a borrower, allowing for the identification of the borrower through their ID and classification by their type. The `borrowedBooks` vector enables tracking the books borrowed by the borrower. This class serves as a foundation for managing borrower-related functionality, such as adding, removing, or retrieving borrowed books.

3.1 Getter methods:

The `getId()` method returns the ID of the borrower, which is of type `int`. It allows external code to retrieve the unique identifier assigned to the borrower. The `getType()` method returns the type or category of the borrower as a string. This method enables the retrieval of the classification of the borrower, which can be useful for distinguishing between different types of borrowers, such as students, faculty, or members of a specific group. The `getBorrowedBooks()` method returns a `vector<string>` containing the titles of the books borrowed by the borrower. It provides access to the list of books borrowed by the borrower, allowing external code to retrieve and utilize this information. These getter methods provide a way to obtain the borrower's ID, type, and the titles of the books they have borrowed. By utilizing these methods, external code can retrieve the relevant information about a borrower without directly accessing the private data members of the Borrower class.

3.2 Setter methods:

The `setId()` method allows the borrower's ID to be set or updated. It takes an `int` parameter and assigns it to the `id` member variable.

The `setType()` method enables the borrower's type to be set or modified. It takes a `string` parameter and assigns it to the `type` member variable.

The `borrowBook()` method adds a book to the list of borrowed books. It takes a `string` parameter representing the book ID and appends it to the `borrowedBooks` vector.

The `setBorrowedBooks()` method allows the entire list of borrowed books to be set or updated. It takes a `vector<string>` parameter and assigns it to the `borrowedBooks` member variable.

The `returnBook()` method removes a specific book from the list of borrowed books. It takes a `string` parameter representing the book ID and removes it from the `borrowedBooks` vector.

The implementation of the `returnBook()` method is responsible for finding and removing a specific book from the `borrowedBooks` vector. The code uses a loop to iterate over the elements of the `borrowedBooks` vector. It compares each element to the provided `bookId` using the `==` operator. If a match is found, indicating that the borrower has borrowed the specified book, the code removes the element from the vector using the `erase()` function. After removing the book, the method returns, indicating that the book has been successfully returned. This implementation ensures that the book is properly removed from the list of borrowed books, allowing the borrower to return it. It helps maintain an accurate record of the books currently borrowed by the borrower.

These setter methods provide a way to modify the borrower's ID, type, and the list of borrowed books. By utilizing these methods, external code can update the relevant information of a borrower and manage their borrowed books.

The `hasBorrowedBook()` method checks whether the borrower has already borrowed a book with the specified `bookId`. It uses the `std::find()` algorithm to search for the `bookId` within the `borrowedBooks` vector. If the `bookId` is found, the method returns `true`; otherwise, it returns `false`. This method helps to determine if the borrower has already borrowed a specific book.

The `addBorrowedBook()` method is used to add a new book with the given `bookId` to the borrower's list of borrowed books. Before adding the book, it first checks if the borrower has already borrowed the same book by calling the `hasBorrowedBook()` method. If the borrower has not borrowed the book yet, it is added to the `borrowedBooks` vector using the `push_back()` function. This method prevents adding duplicate entries to the list of borrowed books. Together, these methods provide functionality for checking if a book has already been borrowed by the borrower and for adding new books to the borrower's list of borrowed books, ensuring data consistency and preventing duplicate entries.

`removeBorrowedBook()` function is responsible for removing a book with the specified `bookId` from the `borrowedBooks` vector. First, it uses the `std::find()` algorithm to search for the `bookId` within the `borrowedBooks` vector. The returned iterator points to the position of the `bookId` if it is found in the vector, or it points to the `borrowedBooks.end()` iterator if the `bookId` is not present. Next, the function checks if the iterator is not equal to `borrowedBooks.end()`, which indicates that the `bookId` was found in the vector. In that case, the `erase()` function is called on the iterator to remove the book from the vector. By using this function, you can remove a borrowed book from the borrower's list of borrowed books by providing its `bookId`. This helps in managing the borrower's book inventory and keeping track of borrowed books.

`display()` function within the `Borrower` class is responsible for displaying the borrower's information, including their ID, type, and the list of borrowed books. First, the function outputs the borrower's ID and type using the `cout` statement. Then, it checks if the `borrowedBooks` vector is empty. If it is empty, it displays "None" to indicate that the borrower has not borrowed any books. If the `borrowedBooks` vector is not empty, the function enters the `else` block and iterates over each `bookId` in the `borrowedBooks` vector. For each `bookId`, it outputs the book ID using the `cout` statement. By using this function, you can easily display the borrower's information, including their borrowed books, providing a clear overview of their borrowing status.

4. Library Class:

The `Library` class encapsulates the functionality of a library and maintains three private member variables: `books`, `borrowers`, and `users`. The `books` vector stores instances of the `Book` class, representing the collection of books available in the library. Each book object contains essential information like the title, author, ISBN, and the number of available copies. The `borrowers` vector contains objects of the `Borrower` class, representing individuals who borrow books from the library. Each borrower is associated with a unique ID and may have a list of borrowed books. The `users` vector holds objects of the `User` class, representing registered users of the library system. These users can include librarians, administrators, or regular patrons, each with different roles and privileges within the library. By organizing books, borrowers, and users in separate vectors, the `Library` class facilitates efficient management of the library's resources and user interactions, allowing for operations like adding new books, registering borrowers, and maintaining user information within the library system.

4.1 Load All information & Data from Text file:

The load function in the Library class serves the purpose of loading all the information and data from text files into the library's collections of books, borrowers, and users. It takes three parameters: `bookFile`, `borrowerFile`, and `userFile`, which represent the paths or names of the text files containing the respective data. By invoking three separate functions within the load function, namely `loadBooks`, `loadBorrowers`, and `loadUsers`, the process of loading data from the text files is distributed into individual tasks. Each of these functions handles the responsibility of reading the corresponding text file and populating the appropriate vectors in the Library class. This modular approach enables better code organization and enhances maintainability. Ultimately, the load function streamlines the loading process, ensuring that the library's collections are initialized with the necessary data for smooth operation.

4.2 Load Book Data:

The `loadBooks` function is responsible for reading and parsing data from a text file containing book information. It takes a parameter `bookFile`, which represents the path or name of the book data file. The function begins by opening the file using an input file stream (`ifstream`). If the file fails to open, an error message is displayed, and the function returns. Next, the function reads each line of the file using `getline` and processes it to extract individual fields such as title, author, ISBN, type, available copies, and reserved IDs. These fields are separated by commas in the file. The extracted data is stored in their respective variables. The available copies are converted from a string to an integer using `stoi`, and the reserved IDs are stored in a vector of integers. If the reserved IDs field is not empty, the function further processes it by extracting individual IDs and converting them to integers. For each line of data, a `Book` object is created and initialized with the extracted information. The reserved IDs are set using the `setReservedIds` function of the `Book` class. Finally, the book object is added to the `books` vector in the Library class. Once all the lines in the file have been processed, the file is closed. The `loadBooks` function effectively reads the book data file, extracts the relevant information, creates `Book` objects, and populates the `books` vector in the Library class with the parsed book data.

4.3 Load Borrowers Information:

The `loadBorrowers` function is responsible for reading and parsing data from a text file containing borrower information. It takes a parameter `borrowerFile`, which represents the path or name of the borrower data file. Similar to the previous `loadBooks` function, it starts by opening the file using an input file stream (`ifstream`). If the file fails to open, an error message is displayed, and the function returns. The function reads each line of the file using `getline` and processes it to extract individual fields such as borrower ID, borrower type, and borrowed books. These fields are separated by commas in the file. The extracted data is stored in their respective variables. The borrower ID is converted from a string to an integer using `stoi`, and the borrowed books are stored in a vector of strings. If the borrowed books field is not empty, the function further processes it by extracting individual book IDs and adding them to the vector. For each line of data, a `Borrower` object is created and initialized with the extracted information. The borrowed books are set using the `setBorrowedBooks` function of the `Borrower` class. Finally, the borrower object is added to the `borrowers` vector in the Library class. Once all the lines in the file have been processed, the file is closed. The `loadBorrowers` function effectively reads the borrower data file, extracts the relevant information, creates `Borrower` objects, and populates the `borrowers` vector in the Library class with the parsed borrower data.

4.4 Add User Information:

The `addUser` function allows the addition of user information to the library system. It prompts the user to enter details such as user ID, user name, user type, and user password. After gathering the input, it checks if the user ID already exists by searching the users vector using the `find_if` algorithm. If a user with the same ID is found, an error message is displayed, and the function returns. Next, the function asks the user to enter the user name and validates the user type. It uses a do-while loop to repeatedly prompt for the user type until a valid value is entered. Valid user types are "student", "teacher", or "researcher". If an invalid user type is provided, an error message is displayed, and the loop continues. Once a valid user type is entered, the loop is exited. The function then prompts the user to enter the user password. After gathering all the required information, a new User object is created and initialized with the provided data. The User object is then added to the users vector in the Library class. Finally, a success message is displayed, indicating that the new user has been added successfully to the library system. The `addUser` function facilitates the collection of user details, performs necessary validations, creates a new User object, and adds it to the users vector, effectively expanding the user database in the library system.

4.5 Modify User Information:

The `modifyuser()` function is designed to modify the name of a user within a collection. It starts by asking the user to enter the user ID they want to modify. Once the ID is obtained, the function searches for a matching ID within the collection of users. If a match is found, the program prompts the user to input a new name for the user. It utilizes `cin.ignore()` to clear the input buffer, ensuring smooth input of the name. The `getline(cin, username)` function is then used to read the entire line of text entered by the user, including any spaces, and stores it in the `username` variable. The `setName()` method of the corresponding User object is called, updating the user's name to the newly entered name. The program sets the boolean variable `found` to `true` to indicate that a user with the given ID was successfully found and modified. Finally, the program notifies the user with the message "user Name modified successfully." to confirm the successful modification. If a matching user is found and modified, the program exits the loop using the `break` statement since the necessary modification has been completed. Once the loop finishes iterating through all the users or if no matching user is found, the `modifyuser()` function comes to an end.

4.6 Add User Information:

The `deleteuser()` function is used to remove a user from a collection of users based on their ID. It looks for the user in the collection by comparing their ID with each user's ID. If a matching user is found, they are removed from the collection. The function then displays a message confirming the successful deletion. If no matching user is found, the function lets you know that the user with the provided ID was not found in the collection. In simple terms, the function deletes a user if they exist in the collection and informs you whether the deletion was successful or not.

4.7 Load User Data:

The `loadUsers` function is responsible for loading user data from a text file and populating the users vector in the library system. It first attempts to open the specified user file using an `ifstream` and checks if the file is successfully opened. If the file fails to open, an error message is displayed, and the function returns. Using a while loop, the function reads each line of the file and extracts the user data by creating a stringstream from the line. The individual values are then retrieved from the stringstream using `getline` and stored in corresponding variables. The ID, initially in string format, is converted to an integer using `stoi`. With the extracted data,

a new User object is created and initialized with the ID, name, type, and password. This newly created User object is then added to the users vector using the `push_back` function, effectively populating the vector with the user information read from the file. Once all lines in the file have been processed, the function closes the file. This ensures that the user data is loaded and available for use within the library system.

4.8 Save Books:

The `saveBooks` function is responsible for saving the book data from the books vector into a text file. It first attempts to open the specified book file using an `ofstream` and checks if the file is successfully opened. If the file fails to open, an error message is displayed, and the function returns. Using a `for` loop, the function iterates over each book in the books vector. For each book, it retrieves the necessary information such as title, author, ISBN, type, and available copies using appropriate getter methods. These values are then written to the file, separated by commas, using the `<<` operator. Next, the function retrieves the reserved IDs for the book using the `getReservedIds` method. If the vector of reserved IDs is not empty, the function iterates over the IDs and writes them to the file, separated by commas. The last reserved ID is handled separately to avoid appending an extra comma. After writing the book and its reserved IDs, a new line character is added to separate each book's data in the file. Once all books have been processed, the function closes the file, ensuring that the book data is saved and stored in the specified file location.

4.9 Save Borrowers:

The `saveBorrowers` function is responsible for saving the borrower data from the borrowers vector into a text file. It first attempts to open the specified borrower file using an `ofstream` and checks if the file is successfully opened. If the file fails to open, an error message is displayed, and the function returns. Using a `for` loop, the function iterates over each borrower in the borrowers vector. For each borrower, it retrieves the borrower's ID and type using appropriate getter methods. These values are then written to the file, separated by commas, using the `<<` operator. Next, the function retrieves the borrowed books for the borrower using the `getBorrowedBooks` method. If the vector of borrowed books is not empty, the function iterates over the books and writes them to the file, separated by commas. The last borrowed book is handled separately to avoid appending an extra comma. After writing the borrower's ID, type, and borrowed books, a new line character is added to separate each borrower's data in the file. Once all borrowers have been processed, the function closes the file, ensuring that the borrower data is saved and stored in the specified file location.

4.10 Save User:

The `saveUsers` function is responsible for saving the user data from the users vector into a text file. It first attempts to open the specified user file using an `ofstream` and checks if the file is successfully opened. If the file fails to open, an error message is displayed, and the function returns. Using a `for` loop, the function iterates over each user in the users vector. For each user, it retrieves the user's ID, name, type, and password using appropriate getter methods. These values are then written to the file, separated by commas, using the `<<` operator. After writing the user's data, a new line character is added to separate each user's data in the file. Once all users have been processed, the function closes the file, ensuring that the user data is saved and stored in the specified file location.

Save data on text file: The save function is responsible for saving all the data of the library system into separate text files. It takes three parameters: `bookFile`, `borrowerFile`, and `userFile`, which represent the file paths for saving book data, borrower data, and user data, respectively. Inside the function, it calls three separate functions: `saveBooks`,

`saveBorrowers`, and `saveUsers`. These functions are responsible for saving the corresponding data into their respective files. By calling these functions one after another, the save function ensures that all the data is saved in the specified files. Each individual save function (`saveBooks`, `saveBorrowers`, and `saveUsers`) follows a similar pattern of opening the respective file, iterating over the corresponding data structures (books, borrowers, and users), and writing the data into the file according to a specific format. By using the save function, all the data of the library system can be conveniently saved into separate text files, allowing for easy retrieval and preservation of the system's state.

4.11 Book Delete Method:

The `deleteBook` method is used to remove a book from the library's collection based on its ISBN (International Standard Book Number). The method takes the ISBN of the book as a parameter. To delete the book, the method searches for it within the books vector using the `std::find_if` algorithm. The lambda function provided to `std::find_if` compares the ISBN of each book in the vector with the given ISBN until a match is found. If a book with the specified ISBN is found (`it != books.end()`), it is removed from the vector using the `erase` function, effectively deleting it from the library. A confirmation message is then displayed, indicating that the book with the given ISBN has been successfully deleted. If no book with the specified ISBN is found, a message is displayed stating that the book was not found in the library's collection. This `deleteBook` method enables the library system to remove books based on their unique ISBN, allowing for efficient management of the book inventory and ensuring accurate records of available books.

4.12 Insert Book:

The `insertBook` function is responsible for adding a new book to the library's collection. It prompts the user to enter the details of the book, such as the title, author, ISBN (International Standard Book Number), type, and the number of available copies. The function starts by displaying a series of prompts to guide the user in providing the book details. It uses the `getline` function to read string inputs for the title, author, ISBN, and type. The number of available copies is read using `cin`. After obtaining all the necessary information, this Book object represents the newly added book. The newly created Book object is then added to the books vector, which holds the collection of books in the library. By adding the book to this vector, it becomes part of the library's inventory. Finally, a message is displayed to confirm that the book has been successfully inserted into the library's collection. The `insertBook` function allows library staff or administrators to easily add new books to the library's catalog. It ensures that the book information is accurately captured and stored, making it available for borrowing by library users.

4.13 Book Modify:

The `modifyBook` function allows the user to modify the details of a book in the library's collection. It takes the ISBN of the book as a parameter to identify the specific book that needs to be modified. The function begins by searching for the book with the provided ISBN in the books vector using the `find_if` algorithm. If a book with the matching ISBN is found, the function proceeds with the modification. The user is then prompted to enter the new details for the book, including the title, author, type, and the number of available copies. Each detail is obtained using the `getline` and `cin` functions. After retrieving the new information, the book object pointed to by the iterator it is updated with the new values using the appropriate setter methods. The title, author, type, and available copies are modified accordingly. A message is displayed to confirm that the book has been successfully modified. If no book with the provided ISBN is found in the library's collection, an appropriate message is displayed to indicate that

the book was not found. The `modifyBook` function provides a way to update the information of a book in the library. It allows for corrections, updates, or changes to the book's details, ensuring that the library's catalog remains accurate and up to date.

4.14 Search Book:

The `searchBook` function allows the user to search for books in the library's collection based on either the book name or the ISBN (International Standard Book Number). The function prompts the user to enter the search type, which can be either "name" or "ISBN". It also prompts the user to enter the search term, which corresponds to the book name or the ISBN number. Based on the search type and search term provided by the user, the function performs the search operation. If the search type is "name", the function iterates through each book in the `books` vector and checks if the book's title matches the search term. If a match is found, the book is added to the `foundBooks` vector. If the search type is "ISBN", the function uses the `find_if` algorithm to search for a book in the `books` vector with a matching ISBN. If a book is found, it is added to the `foundBooks` vector. After completing the search operation, the function checks if any books were found. If `foundBooks` is not empty, it displays the number of books found and then proceeds to display the details of each book using the `display` method of the `Book` class. If no books are found, a message is displayed indicating that no books match the search criteria. The `searchBook` function provides a convenient way for users to find specific books in the library's collection based on their names or ISBN numbers, assisting them in locating the desired books efficiently.

4.15 Delete Borrower:

The `deleteBorrower` function allows the removal of a borrower from the library's list of borrowers based on their borrower ID. The function takes an integer parameter `borrowerId`, which represents the ID of the borrower to be deleted. It uses the `find_if` algorithm to search for a borrower in the `borrowers` vector with a matching ID. If a borrower with the specified ID is found (it is not equal to `borrowers.end()`), the function removes the borrower from the `borrowers` vector using the `erase` method. It then displays a message confirming the deletion of the borrower. If no borrower with the specified ID is found, the function displays a message indicating that the borrower was not found in the list. The `deleteBorrower` function provides a way to remove borrowers from the library's records when needed, ensuring that the borrower information remains accurate and up to date.

4.16 Modify Borrower:

The `modifyBorrower` function allows the administrator or librarian to modify the borrower type for a specific borrower. The user is prompted to enter the borrower ID they wish to modify. The function then iterates through the `borrowers` vector to find a borrower with a matching ID. If a match is found, the user is prompted to enter the new borrower type. The newline character is ignored to prevent any issues with subsequent input. The borrower's type is updated with the new value using the `setType` method. A success message is displayed, indicating that the borrower has been modified successfully. If no borrower is found with the provided ID, a message is displayed indicating that the borrower was not found. This function ensures efficient management of borrower information, allowing for updates to be made as needed.

4.17 Insert Borrower:

The `insertBorrower` function facilitates the addition of a new borrower to the library's borrower list. It prompts the user to provide the borrower's ID and type. The ID is read as an integer using `cin`, and any newline characters are ignored. The borrower's type is obtained as a

string using `getline`, allowing for input with spaces. A new `Borrower` object is created using the provided ID and type, and it is appended to the `borrowers` vector using `push_back`. Lastly, a success message is displayed, indicating the successful insertion of the borrower into the library's records. This function enables administrators or librarians to seamlessly incorporate new borrowers into the library system, ensuring accurate borrower management.

4.18 Search Borrower By ID:

The `searchBorrowerByID` function allows the user to search for a borrower by their ID. The function takes the borrower ID as input and searches for a matching ID in the `borrowers` vector. Using the `find_if` algorithm, it iterates through the vector to find a borrower with the specified ID. If a match is found, the borrower's information is displayed, including the ID and type. If the borrower has borrowed any books, the book IDs are also displayed. However, if the borrower has not borrowed any books, a message indicating that no books are currently borrowed is displayed. If no borrower is found with the provided ID, a message is displayed indicating that the borrower was not found. This function facilitates easy retrieval of borrower information based on their ID, providing a quick overview of their borrowing status and details.

4.19 Check User Login:

The `checkUserLogin` function is used to verify user login credentials. It takes an `userID` and `userType` as input and checks if a matching user exists in the `users` vector. Using the `find_if` algorithm, it searches for a user whose ID and type match the provided credentials. If a matching user is found, the function outputs a message indicating a successful login. However, if no user is found with the provided credentials, or if the credentials are incorrect, a message stating that the user was not found or that the credentials are incorrect is displayed. This function allows for the validation of user login attempts, providing feedback on the success or failure of the login process based on the provided credentials.

4.20 Reserve Book:

The `reserveBook` function is responsible for reserving a book for a specific user. It starts by searching for the book in the `books` vector using the `find_if` algorithm and a lambda function. The lambda function compares the ISBN of each book with the provided ISBN to find a match. If a book with a matching ISBN is found, the function proceeds.

Next, the function attempts to find the user in the `users` vector by using the `find_if` algorithm with another lambda function. This lambda function compares the user ID of each user with the provided `userID`. If a user with a matching ID is found, the function continues.

Once both the book and the user are found, the function checks if the book is available for reservation by calling the `isAvailable()` function on the book object. If the book is available, the user's ID is added to the reserved list of the book using the `addReservedId()` function.

Finally, the function displays a message indicating whether the reservation was successful or not. If the book was successfully reserved, the message informs the user that the reservation was successful. Otherwise, if the book was not available for reservation or if either the book or the user was not found, the message notifies the user that the reservation could not be made.

4.21 Borrow Book:

The `borrowBook` function is responsible for borrowing a book by a specific user. It begins by searching for the book in the `books` vector using the `find_if` algorithm and a lambda function. The lambda function compares the ISBN of each book with the provided ISBN to find a match. If a book with a matching ISBN is found, the function proceeds.

Next, the function attempts to find the borrower in the borrowers vector by using the `find_if` algorithm with another lambda function. This lambda function compares the user ID of each borrower with the provided `userID`. If a borrower with a matching ID is found, the function continues.

Once both the book and the borrower are found, the function checks if the book is available for borrowing by calling the `getAvailableCopies()` function on the book object. If there are available copies, the function decreases the available copies of the book by one using the `setAvailableCopies()` function.

The function then checks if the user has already borrowed the book by calling the `hasBorrowedBook()` function on the borrower object. If the user has already borrowed the book, the function adds the book to the borrower's borrowed books vector using the `addBorrowedBook()` function and displays a success message.

If the user is a new borrower, the function also adds the book to the borrower's borrowed books vector and displays a success message indicating that the user is a new borrower.

If there are no available copies of the book, the function displays a message informing the user that there are no available copies.

If either the book or the borrower is not found, the function displays a corresponding error message.

4.22 Return Book:

The `returnBook` function is responsible for returning a borrowed book by a specific user. It begins by searching for the borrower in the borrowers vector using the `find_if` algorithm and a lambda function. The lambda function compares the user ID of each borrower with the provided `userID` to find a match. If a borrower with a matching ID is found, the function proceeds.

Once the borrower is found, the function displays the borrowed books by calling the `display()` function on the borrower object. The user is then prompted to enter the ID of the book they want to return.

The function checks if the borrower has borrowed the book with the provided book ID by calling the `hasBorrowedBook()` function on the borrower object. If the borrower has borrowed the book, a success message is displayed.

If the borrower has not borrowed the book, an error message is displayed, and the function returns, indicating that the book cannot be returned.

If the borrower has borrowed the book, the function removes the book from the borrower's borrowed books by calling the `returnBook()` function on the borrower object.

Finally, a message is displayed indicating the successful return of the book by displaying the book ID and the user ID.

If the borrower is not found, an error message is displayed, indicating that the provided user ID is invalid.

4.23 Check Password & login:

The `checkUserPresence` function is responsible for checking the presence of a user based on their user ID, user type, and password. It begins by searching for the user in the users vector using the `find_if` algorithm and a lambda function. The lambda function compares the user ID and user type of each user with the provided `userID` and `userType` to find a match. If a user with a matching ID and type is not found or the password does not match the user's password, an error message is displayed indicating that the user with the provided ID, type, and password is not valid. The function returns false to indicate the presence check failure. If the user is found and the password matches, a success message is displayed, including the user's name. The function returns true to indicate the presence check success.

4.24 Main Function:

Main function of a library management system allows users to interact with the library system either as an admin or a user. Below is a breakdown of how the code works:

The Library object is created.

The load function is called on the library object to load data from three files: "books.txt", "borrowers.txt", and "users.txt".

The user is prompted to enter their role as either an "admin" or a "user".

If the user selects "admin", they are prompted to enter an admin password. If the password is correct, an admin menu is displayed with several options.

Depending on the admin's choice, functions like `insertBook`, `modifyBook`, `deleteBook`, `searchBook`, `insertBorrower`, `modifyBorrower`, `deleteBorrower`, or `save` are called on the library object.

If the user selects "user", they are prompted to either create a new account or provide their ID, user type, and password.

If the user chooses to create a new account, the `addUser` function is called on the library object.

If the user chooses to enter their ID, user type, and password, the `checkUserPresence` function is called to verify their credentials.

If the user's credentials are valid, a user menu is displayed with options such as `borrowBook`, `reserveBook`, `returnBook`, `searchBook`, or `save`.

Depending on the user's choice, corresponding functions are called on the library object.

The program continues to execute until the user chooses to exit.

Overall, the program provides a basic menu-driven interface for managing a library system. However, without the implementation of the Library class and its member functions, it's difficult to provide further insights into the functionality of the system.

Model Discussions:

1) Pre-Implementation Assumptions:

Before implementing the library management system, several pre-implementation assumptions are made. These assumptions include having specific details for each book, such as title, author, ISBN, genre, and available copies. The system assumes a distinction between users and borrowers, where users can search for books and borrowers are registered users with borrowing privileges. Unique identifiers, such as ISBN for books and user ID for borrowers, are assumed for accurate identification and tracking. The system also assumes the presence of book reservation functionality to allow users to reserve copies. Error handling mechanisms are assumed to handle invalid inputs, duplicate entries, and borrower management. The implementation assumes a console-based interface for user interaction, involving command-line inputs and output messages. Communicating and validating these assumptions with stakeholders ensures a shared understanding of the project's scope and functionality, leading to an effective implementation process. To ensure successful implementation, it is essential to communicate and validate these assumptions with relevant stakeholders. This helps establish a shared understanding of the project's scope and functionality, allowing for effective collaboration and a more seamless development process. By aligning expectations and requirements, the library management system can be implemented in a way that meets the needs of the users and enhances the overall library experience.

Model:

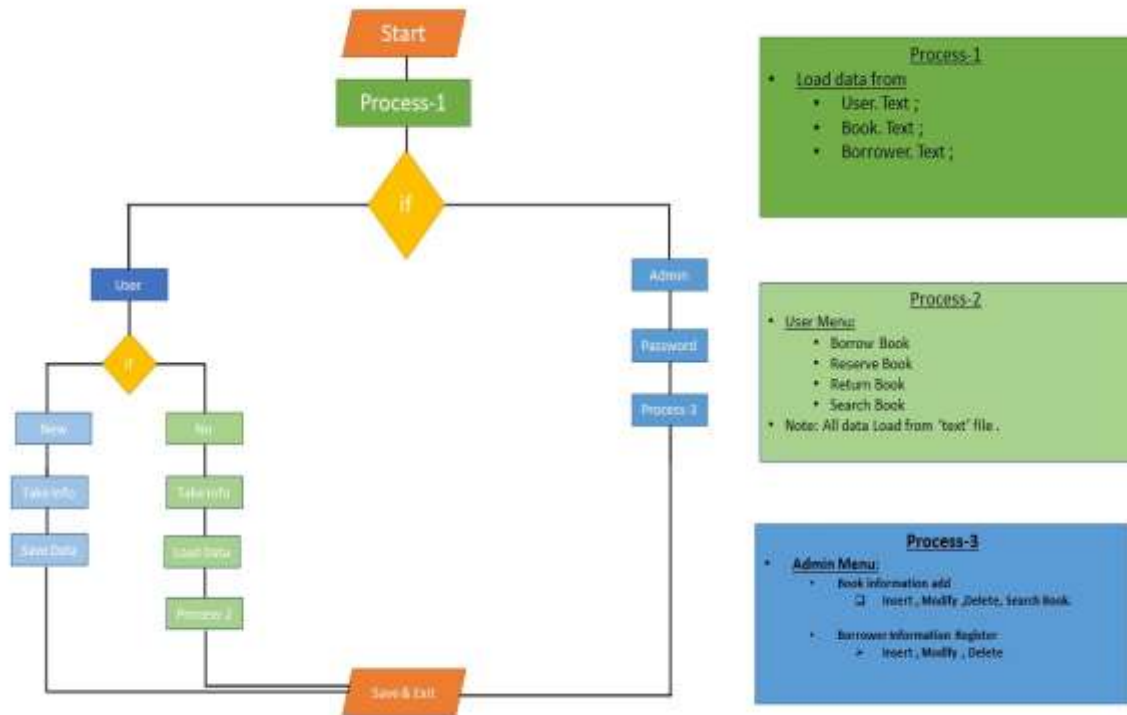


Figure 1 Flowchart of Library Management System

2) Program Functionalities:

• User Interface

The code implements a menu-driven interface for both admins and users. Depending on the user's choice, different functions are called on the library object. This approach provides a structured way for users to interact with the library system.

• Book modify

The `modifyBook()` function implements the algorithm for modifying a book in the library based on its ISBN. The algorithm utilizes the `std::find_if` function from the C++ Standard Library to search for the book with the specified ISBN in the books vector of the Library object. The algorithm starts by searching for the book in the books vector using the ISBN provided as a parameter. It uses a lambda function as the predicate for the search, comparing the ISBN of each book with the target ISBN. If a book with the given ISBN is found, the algorithm proceeds to prompt the user for new details of the book. The user is asked to enter the updated title, author, type, and the number of available copies. Once the new information is collected, the algorithm updates the corresponding fields of the book object found in the vector. It uses the iterator `it` to access the book and modifies its title, author, type, and available copies accordingly. Finally, the algorithm provides feedback to the user, indicating whether the book modification was successful. If a book with the specified ISBN is not found in the vector, an appropriate message is displayed. By employing this algorithm, the `modifyBook()` function enables the user to update the information of a book in the library's collection. It utilizes the vector data structure to efficiently locate the book for modification, ensuring accurate and timely updates to the library's book records.

- Book Insert

The `insertBook()` function implements the algorithm for inserting a book into the library. It follows a step-by-step process to gather the necessary book information from the user, create a new Book object, and add it to the books vector in the Library object. The algorithm begins by prompting the user to enter the required book details, such as the title, author, ISBN, and other relevant information. Once the user provides the input, a new Book object is created to store this information. Next, the newly created Book object is added to the books vector, which represents the collection of books in the library. This vector serves as a dynamic data structure that can grow or shrink as books are added or removed from the library. By utilizing the books vector, the algorithm ensures efficient storage and retrieval of book records. It allows for easy management of the library's book collection, including operations like searching, modification, and deletion. Overall, the `insertBook()` function employs the book insertion algorithm to facilitate the addition of new books to the library. By following this algorithmic process, the function ensures that book records are properly created and integrated into the library's data structure, enabling effective organization and access to the books.

- Book Borrow

The `borrowBook()` function implements the algorithm for borrowing a book from the library. It takes the ISBN of the book and the user ID as parameters. The algorithm begins by searching for the book with the given ISBN in the books vector using the `std::find_if` function. If the book is found, the algorithm proceeds to search for the borrower with the provided user ID in the borrowers vector.

If both the book and borrower are found, the algorithm continues to borrow the book. First, it checks the availability of copies for the book. If there are available copies, the algorithm decreases the number of available copies by one. Then, it checks whether the borrower has already borrowed the same book before. If so, it updates the borrower's borrowed books record and displays a success message indicating that the book has been borrowed successfully. Otherwise, if the user is a new borrower, the book is added to their borrowed books vector, and a message is displayed confirming the successful borrowing. If the book is not available (no available copies), the algorithm informs the user that there are no copies of the book currently available. If the borrower or the book is not found in their respective vectors, appropriate error messages are displayed. By utilizing this algorithm, the `borrowBook()` function enables a user to borrow a book from the library's collection. It employs searching techniques using `find_if` to efficiently locate the book and borrower, and utilizes vector data structures for book and borrower storage, ensuring effective book borrowing operations and accurate book availability tracking.

- Book Delete

The `deleteBook()` function implements the algorithm for deleting a book from the library's collection. It takes the ISBN of the book to be deleted as a parameter. The algorithm begins by searching for the book with the specified ISBN in the books vector using the `std::find_if` function. If the book is found, the algorithm proceeds to delete it from the vector. If the book is found in the vector, the algorithm uses the `erase()` function to remove the book from the vector, effectively deleting it. A success message is then displayed, confirming that the book with the specified ISBN has been deleted. On the other hand, if the book is not found in the vector, the algorithm displays an error message indicating that the book with the specified ISBN was not found. By employing this algorithm, the `deleteBook()` function provides the functionality to remove a book from the library's collection. It utilizes the `find_if` algorithm to

efficiently locate the book and the `erase()` function to remove it from the vector. The use of vectors as the data structure for storing books ensures efficient deletion operations, allowing for effective management of the library's book collection.

- Book Reserve

The `reserveBook()` function implements the algorithm for reserving a book in the library. It takes the ISBN of the book and the user ID as parameters. The algorithm begins by searching for the book with the specified ISBN in the books vector using the `std::find_if` function. Similarly, it searches for the user with the specified ID in the users vector. If both the book and user are found, the algorithm proceeds to reserve the book for the user. Upon finding the book and user, the algorithm adds the user's ID to the reserved IDs of the book using the `addReservedId()` method. This indicates that the book has been reserved by the user. A success message is then displayed, confirming the reservation with the book's ISBN and the user's ID.

- Book Search

The `searchBook()` function implements the algorithm for searching books in the library based on user-specified criteria. The algorithm prompts the user to enter search criteria such as title, author, or genre. It then iterates over the books vector and compares each book's attributes with the search criteria. Matching books are collected and displayed to the user. This algorithm provides a way for users to find books in the library that match their desired criteria.

3) Relation with Data structure and Algorithm

```
int main() {
    Library library;
    library.load("books.txt", "borrowers.txt", "users.txt");
    The Library class is instantiated which helps to organize and manage books, borrowers, and
    users it includes appropriate data members and member functions to handle library operations.
    The load function is called to load data from external files ("books.txt", "borrowers.txt", and
    "users.txt") into the library object. Using file I/O to store and retrieve library data. string
    userType;
    bool correction = false;
    do {
        cout
        *****
    endl;
        cout << "***** Welcome to our Library
        *****" << endl;
        cout
        *****
    endl;
        cout << "->> Please Select Your Choice " << endl;
        cout << "->> Want to enter as (a user) or (an admin)? ";
        getline(cin, userType);
    The user is prompted to select their user type (admin or user) and their choice is stored in the
    userType variable.
    if (userType == "admin") {
        correction = true;
        string adminPassword;
```

```

cout << "Enter admin password: ";
cin >> adminPassword;
cin.ignore();
// Check if the admin password is correct
if (adminPassword == "admin123") {
    string adminChoice;
    while (true) {
        cout << "Admin Menu:" << endl;
        // Display various admin menu options
        ...
    }
}
else {
    cout << "Invalid admin password. Access denied." << endl;
}
}

```

If the user selects "admin" as their user type, they are prompted to enter an admin password. Here, the password check algorithm is used to verify if the entered password matches the expected password "admin123". This algorithm involves string comparison to authenticate the admin.

```

else if (userType == "user") {
    correction = true;
    // Check if the user wants to create a new account
    string createAccountChoice;
    cout << "Do you want to create a new account? (yes/no): ";
    cin >> createAccountChoice;
    cin.ignore();
    if (createAccountChoice == "yes") {
        library.addUser();
        correction = false;
    }
    else {
        bool pcheck = false;
        do {
            int userId;
            string userT, userpass;
            cout << "Enter your ID: ";
            cin >> userId;
            cout << "Enter userT (teacher, student, researcher): ";
            cin >> userT;
            cout << "Enter Password: ";
            cin >> userpass;
            cin.ignore();
            if (library.checkUserPresence(userId, userT, userpass)) {
                string userChoice;
                pcheck = true;
                while (true) {
                    cout << "User Menu:" << endl;
                    // Display various user menu options

```

```

        ...
    }
}
} while (pcheck == false);
}
}

```

If the user selects "user" as their user type, they are given options to create a new account or log in with an existing account. Depending on the user's choice, the addUser function or the checkUserPresence function is called. Functions like user objects or user records are related to data structure, which are stored and managed within the Library class. else {

```

    cout << "Invalid choice" << endl;
}

```

This line prints an error message when an invalid user type is entered. The code continues with menu options and function calls for admin and user interactions. The Library class encompasses the necessary data structures and algorithms to handle book, borrower, and user management within the library system.

Data Structures:

Library class: This class represents the library system and contains data structures such as 'arrays', 'vectors', or 'linked lists' to 'store books', 'borrowers', and 'users'. Some of them from library class:

```

void load(const string& bookFile, const string& borrowerFile, const
string& userFile{} ;
void loadUsers(const string& userFile){};
void insertBook(){};
void insertBorrower(){};

```

The project leverages different data structures to store and manage information effectively. The most prominent data structures used include arrays, linked lists, and hash tables. Arrays are employed to store book and borrower details, allowing for quick access and modification. Linked lists facilitate dynamic storage for borrowers, ensuring efficient addition and deletion of user records. Hash tables are utilized to store users' login information, providing fast retrieval and verification of user credentials.

Algorithms:

Several algorithms are integrated into the project to perform essential operations. Linear search is used for book and borrower search, enabling easy retrieval based on attributes like ISBN and ID. String comparison algorithms ensure accurate matching of user input with predefined values, enabling proper flow control within the system. File input/output algorithms are employed to read and write data from external files, ensuring seamless data persistence.

User Authentication: The code prompts the user to enter their ID, user type, and password. The checkUserPresence function uses an algorithm to verify the user's credentials, which could involve searching and comparing user data stored in the library system.

```

    " bool checkUserPresence(int userID, const string& userType,
        const string& password) {
        // Find the user with the given ID and type
        auto userIt = find_if(users.begin(), users.end(), [&](const
User& user) {

```

```

        return user.getId() == userID && user.getType() ==
userType;
    });
    // Check if the user exists and verify the password
    if (userIt == users.end() || userIt->getPassword() !=
password) {
        cout << userType << " with ID " << userID
            << " and provided password is not valid." << endl;
        return false;
    }
    else {
        cout << userIt->getName() << " Login Successful" <<
endl;
        return true; "

```

Menu-Driven Interface: The code implements a menu-driven interface for both admins and users. Depending on the user's choice, different functions are called on the library object. This approach provides a structured way for users to interact with the library system.

```

" cout << "->> Please Select Your Choice " << endl;
  cout << "->> Want to enter as( a user) or (an admin) ? ";
  getline(cin, userType);

  if (userType == "admin") {
      correction = true;
      string adminPassword;
      cout << "Enter admin password: ";
      cin >> adminPassword;
      cin.ignore();"

```

Book and Borrower Operations: The admin menu allows operations like inserting, modifying, and deleting books and borrowers. These operations may involve algorithms such as searching for a book or borrower based on their attributes (e.g., ISBN, ID) and modifying or removing them accordingly.

```

    " cout << "Enter your choice: ";
      getline(cin, adminChoice);

      if (adminChoice == "1") {
          library.insertBook();
      }
      else if (adminChoice == "2") {
          string ISBN;
          cout << "Enter the ISBN of the book to modify: ";
          getline(cin, ISBN);
          library.modifyBook(ISBN);
      }
      else if (adminChoice == "3") {
          string ISBN;
          cout << "Enter the ISBN of the book to delete: ";

```

```

        getline(cin, ISBN);
        library.deleteBook(ISBN);
    }
    else if (adminChoice == "4") {
        library.searchBook();
    }
    else if (adminChoice == "5") {
        library.insertBorrower();
    }
    else if (adminChoice == "6") {
        library.modifyBorrower();”

```

Book Borrowing and Reservation: The user menu includes options for borrowing and reserving books. These operations likely involve algorithms for checking the availability of books, updating their status, and associating them with the respective user or borrower.

```

“void borrowBook(const string& ISBN, int userID) {
    // Find the book with the given ISBN
    auto bookIt = find_if(books.begin(), books.end(), [&](const Book& book) {
        return book.getISBN() == ISBN;
    });

    // Check if the book exists
    if (bookIt != books.end()) {
        Book& book = *bookIt;

        // Find the borrower with the given user ID
        auto borrowerIt = find_if(borrowers.begin(), borrowers.end(), [&](const Borrower& borrower) {
            return borrower.getId() == userID;
        });

        // Check if the borrower exists
        if (borrowerIt != borrowers.end()) {
            Borrower& borrower = *borrowerIt;

            // Decrease the available copies of the book
            int availableCopies = book.getAvailableCopies();
            if (availableCopies > 0) {
                book.setAvailableCopies(availableCopies - 1);

                // Check if the user is already a borrower
                if (borrower.hasBorrowedBook(ISBN)) {
                    borrower.addBorrowedBook(ISBN);
                    cout << "Book borrowed successfully." << endl;
                }
                else {
                    // Add the book to the borrower's borrowed books vector
                    borrower.addBorrowedBook(ISBN);
                    cout << "User is a new borrower. Book borrowed successfully." << endl;
                }
            }
        }
    }
}

```

```
else {  
    cout << "Sorry, no available  
copies of the book." << endl;  
}  
}  
else {  
    cout << "Borrower not found."  
<< endl;  
}
```

File I/O: The load and save functions are responsible for loading data from files into the library system and saving the updated data back to the files. These functions likely use file input/output algorithms for reading and writing data.

```
“void save(const string& bookFile, const string& borrowerFile,
    const string& userFile) {
    saveBooks(bookFile);
    saveBorrowers(borrowerFile);
    saveUsers(userFile);
}”
```

Name of the Algorithm & their uses:

Linear Search:

Usage: Searching for a book or borrower based on their attributes (ISBN, ID) in the searchBook, modifyBook, deleteBook, deleteBorrower, borrowBook, reserveBook, and returnBook functions.

Why: Linear search is a simple algorithm that sequentially checks each element in a data structure until a match is found. It is used here to find specific books or borrowers in the library system.

String Comparison:

Usage: Comparing the user's input (admin password, user type, create account choice) with specific values in the code.

Why: String comparison is used to check if the user's input matches specific strings (e.g., admin password, user type) to control the flow of the program or perform certain actions based on user choices.

File Input/Output:

Usage: Reading data from files in the load function and writing data to files in the save function.

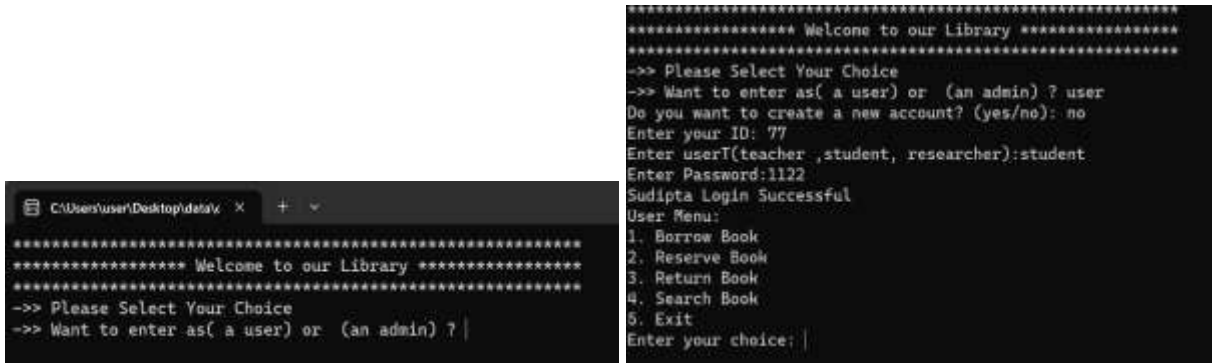
Why: File I/O algorithms are used to read data from external files (e.g., "books.txt", "borrowers.txt", "users.txt") and populate the library system with the data. They are also used to save the updated data back to the files.

Demonstration:

The user gets prompted to choose between four options when running the program. There are four book options: Borrow Book, Reserve Book, Return Book, and a few miscellaneous options: switching to administrator mode, updating the user's information on the system by the admin, exiting the program.

- **Login option; User and Admin panel**

Upon launching the program, a menu will be presented with two choices: User and Admin. When selecting the User option, the program will prompt whether you are a registered user or wish to create a new account. On the other hand, choosing the Admin option will require entering the admin password to gain access to the admin panel, which allows for system information modifications.



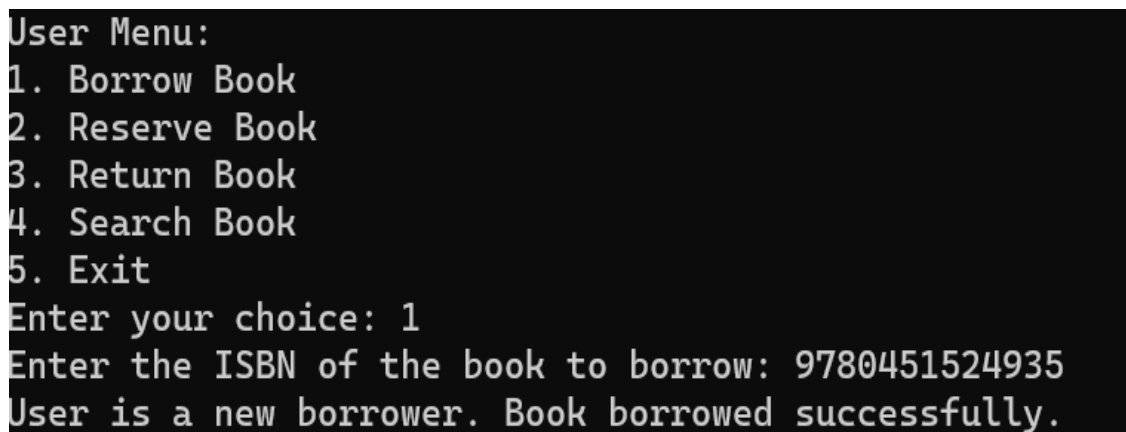
```

***** Welcome to our Library *****
*****
--> Please Select Your Choice
--> Want to enter as( a user) or (an admin) ? user
Do you want to create a new account? (yes/no): no
Enter your ID: 77
Enter userT(teacher ,student, researcher):student
Enter Password:1122
Sudipta Login Successful
User Menu:
1. Borrow Book
2. Reserve Book
3. Return Book
4. Search Book
5. Exit
Enter your choice: |
  
```

Figure 2 User Panel of LMS (Library Management System)

- **Borrow Book option**

Book borrowing in the library management system requires users to obtain approval from the administrator for their borrowing requests. This process ensures that the administrator has reviewed the user's details, such as their name, contact information, and library membership ID. Once the administrator approves the request, the user can proceed with the book borrowing process. To ensure accuracy in identifying the specific edition of the desired book, the user needs to provide the ISBN (International Standard Book Number) of the book they wish to borrow. With the approved request and the provided ISBN, the user can proceed to borrow the book from the library. This ensures that the library keeps track of which books are borrowed and by whom, allowing for better management and accountability. By following this process, users can enjoy their reading experience while ensuring that the library's collection is properly managed and accessible to all users.



```

User Menu:
1. Borrow Book
2. Reserve Book
3. Return Book
4. Search Book
5. Exit
Enter your choice: 1
Enter the ISBN of the book to borrow: 9780451524935
User is a new borrower. Book borrowed successfully.
  
```

Figure 3 Book Borrow Option in Library Management System

- Book-Searching Option

Upon selecting the book-searching option, the user is presented with two choices: searching by ISBN number or by book name. This provides flexibility for the user to search for a specific book using either the unique ISBN number or by its title. Once the user has successfully selected a book, they can access its expanded details, such as additional information or descriptions, to gather more comprehensive information about the book.

```
User Menu:
1. Borrow Book
2. Reserve Book
3. Return Book
4. Search Book
5. Exit
Enter your choice: 4
Enter the search type (name or ISBN): ISBN
Enter the search term (bookname or ISBN#: 9780743273565
Found 1 book(s) matching the search criteria:
Title: The Great Gatsby
Author: F. Scott Fitzgerald
ISBN: 9780743273565
Type: Fiction
Available Copies: Fiction
Reserved ids:
Id number:1
```

Figure 4 Book Searching Option in Library Management System

- Reserve Book

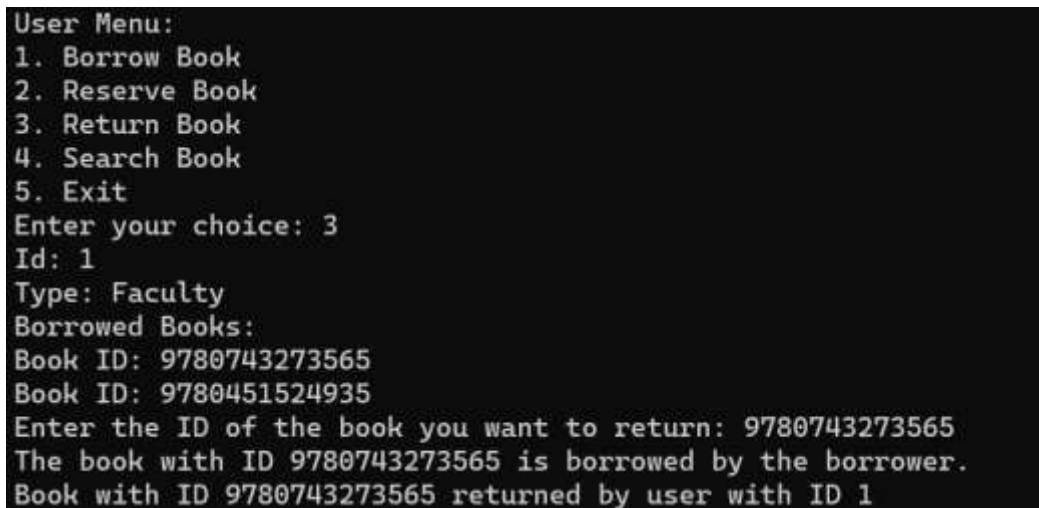
To reserve a book, the system prompts the user to enter the ISBN number of the desired book. Once the user enters the ISBN number, the system automatically reserves the book by associating it with the user's ID. This process ensures that the reserved book is set aside for the user, indicating their intention to borrow it in the future. By linking the ISBN number and the user ID, the system maintains a record of the reservation and ensures that the book is reserved exclusively for the user who initiated the request.

```
User Menu:
1. Borrow Book
2. Reserve Book
3. Return Book
4. Search Book
5. Exit
Enter your choice: 2
Enter the ISBN of the book to reserve: 9780061120084
Book with ISBN 9780061120084 reserved for user with ID 1
User Menu:
1. Borrow Book
2. Reserve Book
3. Return Book
4. Search Book
5. Exit
Enter your choice: |
```

Figure 5 Reserve Book Option in Library Management System

- Return Book

To return a book, the user needs to follow a simple procedure. The program prompts the user to enter their ID to identify themselves. Once the user provides their ID, the program displays a list of books that the user has borrowed. The user is then requested to enter the ID(s) of the book(s) they wish to return. This step allows the user to indicate which specific book(s) they are returning from their borrowed collection. By providing the book ID(s) for return, the user ensures that the returned books are accurately recorded in the system and can be made available for other users to borrow. This process helps maintain the accuracy of book inventory and enables efficient book circulation within the library system.



```

User Menu:
1. Borrow Book
2. Reserve Book
3. Return Book
4. Search Book
5. Exit
Enter your choice: 3
Id: 1
Type: Faculty
Borrowed Books:
Book ID: 9780743273565
Book ID: 9780451524935
Enter the ID of the book you want to return: 9780743273565
The book with ID 9780743273565 is borrowed by the borrower.
Book with ID 9780743273565 returned by user with ID 1
  
```

Figure 6 Return Book Option in Library Management System

- Admin Panel

Once logged in as an administrator, the program presents an admin menu that offers various options for managing the book system. This admin menu includes features such as inserting a new book, modifying an existing book, deleting a book from the system, and searching for books based on specific criteria.

The "insert" option allows the administrator to add a new book to the system. This involves collecting the necessary information about the book, such as its title, author, ISBN, and other details. The new book is then added to the book database, expanding the library collection.

The "modify" option enables the administrator to make changes to the details of an existing book. By selecting this option, the administrator can update the title, author, ISBN, or any other relevant information of the book in the system. This ensures that the book records remain accurate and up to date.

The "delete" option provides the administrator with the capability to remove a book from the system. By specifying the ISBN of the book to be deleted, the administrator can permanently remove it from the book database. This option is useful when a book is no longer available or needs to be removed from the library collection.

The "search" option allows the administrator to find books based on specific criteria, such as title, author, or genre. This functionality helps the administrator locate specific books in the system quickly and efficiently.

Overall, the admin menu provides essential tools for managing and maintaining the book system, empowering the administrator to add, modify, delete, and search for books as needed.

```

->> Please Select Your Choice
->> Want to enter as( a user) or  (an admin) ? admin
Enter admin password: admin123
Admin Menu:
1. Insert Book
2. Modify Book
3. Delete Book
4. Search Book
5. Insert Borrower
6. Modify Borrower
7. Delete Borrower
8. Save and Exit
Enter your choice: |

```

Figure 7 Admin Panel in Library Management System

- Insert book

When the administrator chooses the "insert" option to add a new book, the program initiates a series of questions to gather the necessary information about the book. The administrator is prompted to provide details such as the title of the book, the author's name, the ISBN (International Standard Book Number), the type or genre of the book, and the number of available copies.

The program waits for the administrator's input and collects the information entered for each field. Once all the required information is provided, the program proceeds to create a new book record with the given details. This involves creating an instance of the Book class or data structure, using the provided information to initialize the attributes of the book object.

After successfully creating the book record, the program displays a message to indicate that the book insertion process was successful. This confirmation message lets the administrator know that the book has been added to the system and is now part of the library collection.

This step ensures that the book information is accurately captured and stored in the system, allowing for seamless book management and availability tracking. The successful message provides feedback to the administrator, confirming that the book insertion was completed without any issues.

```

->> Please Select Your Choice
->> Want to enter as( a user) or  (an admin) ? admin
Enter admin password: admin123
Admin Menu:
1. Insert Book
2. Modify Book
3. Delete Book
4. Search Book
5. Insert Borrower
6. Modify Borrower
7. Delete Borrower
8. Save and Exit
Enter your choice: 1
Enter book details:
Title: Ibrahim
Author: frank
ISBN: 1236525987
Type: story
Available Copies: 20
Book inserted successfully.

```

Figure 8 Insert Book Option in Library Management System

- Modify book

When the administrator selects the "modify" option from the admin menu, they gain the ability to make changes to the information of a book. The program prompts the administrator to enter the ISBN (International Standard Book Number) of the book they want to modify.

Once the ISBN is provided, the program searches for the book in the library system based on the given ISBN. If a match is found, the program proceeds to ask the administrator for the updated information of the book. The administrator can modify various details such as the title, author, type or genre, and the number of available copies.

Upon receiving the new information, the program updates the corresponding attributes of the book with the provided values. This involves accessing the book object using its reference or index and modifying the relevant attributes.

After successfully updating the book's information, the program displays a message to confirm that the modification process was successful. This message serves as feedback to the administrator, indicating that the changes have been applied to the book.

By providing the ability to modify book information, the program allows the administrator to keep the library's records accurate and up to date. It enables them to reflect any changes or corrections to the book details, ensuring the integrity of the library catalog.

```
Type: story
Available Copies: 20
Book inserted successfully.
Admin Menu:
1. Insert Book
2. Modify Book
3. Delete Book
4. Search Book
5. Insert Borrower
6. Modify Borrower
7. Delete Borrower
8. Save and Exit
Enter your choice: 2
Enter the ISBN of the book to modify: 234667654587
Enter new book details:
Title: 2024
Author: Sudipta
Type: 60
Available Copies: 30
Book modified successfully.
```

Figure 9 Modify Book Option in Library Management System

- Delete book

When the administrator selects the "delete" option from the admin menu, they are granted the authority to delete book information based on the book's ISBN (International Standard Book Number). The program prompts the administrator to enter the ISBN of the book they want to delete.

Using the provided ISBN, the program searches for the book in the library system. If a book with a matching ISBN is found, the program proceeds to remove that book from the system, effectively deleting its information.

To delete the book, the program utilizes algorithms such as searching and removing elements from a data structure. It may involve iterating over the collection of books, comparing the ISBN of each book with the provided ISBN, and removing the book from the collection if a match is found.

After successfully deleting the book, the program displays a message confirming the deletion. This message serves as feedback to the administrator, indicating that the book with the specified ISBN has been successfully removed from the library system.

By providing the authority to delete book information, the program empowers the administrator to manage the library's catalog and remove any books that are no longer relevant or available. This feature ensures the accuracy and relevance of the library's records and helps maintain an organized and up-to-date collection.

```
Admin Menu:
1. Insert Book
2. Modify Book
3. Delete Book
4. Search Book
5. Insert Borrower
6. Modify Borrower
7. Delete Borrower
8. Save and Exit
Enter your choice: 3
Enter the ISBN of the book to delete: 234667654587
Book with ISBN 234667654587 has been deleted.
```

Figure 10 Delete Book Option on Library Management System

- Insert borrower

The "insert borrower" option in the library management system allows the administrator to add new clients' information into the system. When the admin selects this option, they can enter details such as the client's name, contact information, and other relevant data.

After the admin inserts the borrower's information, it is stored in the system's database. This data structure is designed to efficiently manage and organize borrower records. It is important to note that the borrower's information is initially added to the system and will be subject to approval at a later stage. This means that the system will review and verify the borrower's details before allowing them to borrow books.

The purpose of inserting the borrower's information in advance is to streamline the borrowing process. By having the client's details already in the system, it becomes easier to manage their borrowing requests and track their activity.

The approval process ensures that only authorized borrowers can access the library's resources. It allows the library staff to verify the client's identity, eligibility, and adherence to any borrowing policies.

```
Admin Menu:
1. Insert Book
2. Modify Book
3. Delete Book
4. Search Book
5. Insert Borrower
6. Modify Borrower
7. Delete Borrower
8. Save and Exit
Enter your choice: 5
Enter borrower ID: 22
Enter borrower type: Teacher
Borrower inserted successfully.
```

Figure 11 Insertion Borrower Option on Library Management System

- Modify borrower.

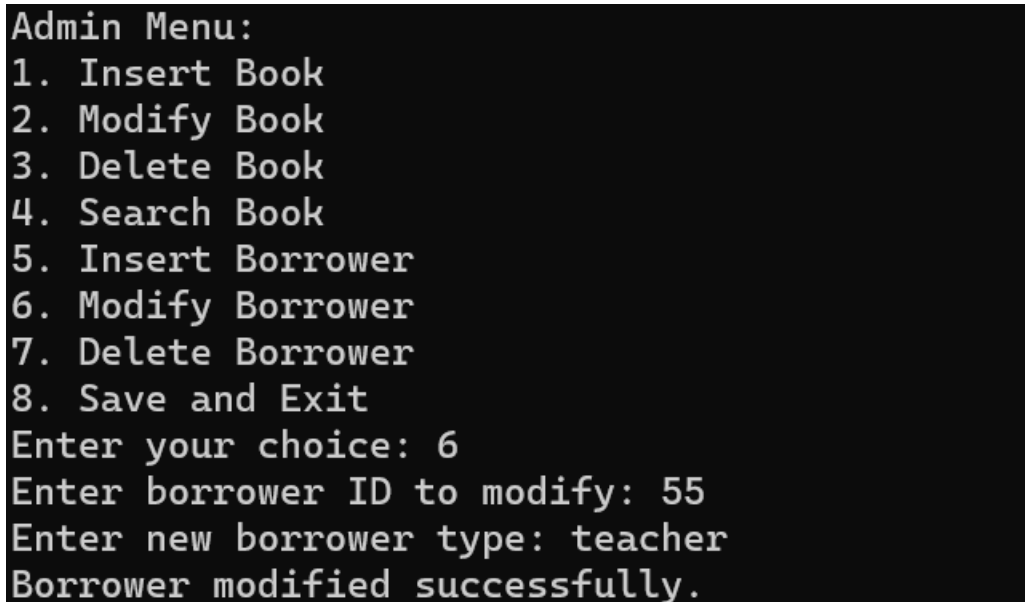
The "modify borrower" option in the library management system provides the admin with the ability to update and modify the information of a borrower if any mistakes or changes occur. This option is designed to ensure the accuracy and integrity of the borrower's data within the system.

When the admin selects the "modify borrower" option, they can access the borrower's information stored in the system's database. This information may include the borrower's name, contact details, and any other relevant data.

By allowing modifications, the admin can correct any errors, update contact information, or make necessary changes to the borrower's record. This ensures that the borrower's information remains up-to-date and accurate.

Modifying borrower information can be useful in situations where a borrower changes their address, phone number, or other personal details. It allows the admin to maintain an accurate record and enables effective communication with borrowers.

Additionally, the "modify borrower" option provides flexibility in case any changes in the borrowing policies or terms of service require an update to the borrower's information. The admin can easily make these modifications to ensure compliance with the latest regulations or library policies.



```
Admin Menu:
1. Insert Book
2. Modify Book
3. Delete Book
4. Search Book
5. Insert Borrower
6. Modify Borrower
7. Delete Borrower
8. Save and Exit
Enter your choice: 6
Enter borrower ID to modify: 55
Enter new borrower type: teacher
Borrower modified successfully.
```

Figure 12 Modify Borrower Option on Library Management System

- Delete borrower

As an administrator, you possess the privilege to remove a borrower from the system by providing their corresponding ID. With this authority, you can effectively delete a borrower's record and associated information. By entering the ID linked to the borrower, you can initiate the deletion process and remove their details from the system. This functionality allows you to manage the borrower database and maintain accurate records. As an administrator, you have the capability to exercise control over the borrower management system and ensure its integrity. By utilizing the ID as a unique identifier, you can confidently delete specific borrowers as needed. This feature provides you with the necessary control and flexibility in managing borrower information.


```

Admin Menu:
1. Insert Book
2. Modify Book
3. Delete Book
4. Search Book
5. Insert Borrower
6. Modify Borrower
7. Delete Borrower
8. Save and Exit
Enter your choice: 7
Enter the ID of the borrower to delete: 55
Borrower with ID 55 has been deleted.

```

Figure 13 Deleting Borrower Option on Library Management System

Conclusion

In conclusion, the Library Management System project serves as a testament to the significance of data structures and algorithms in practical applications. By effectively utilizing various data structures and implementing suitable algorithms, the project ensures streamlined library operations, efficient data storage, and seamless user interaction. It highlights the crucial role of data structures and algorithms in developing robust and scalable software systems.

Reference's

- DelftStack. (2021). Create directory in C++. https://www.delftstack.com/howto/cpp/cpp-create-directory/#use-the-stdfilesystemcreate_directories-function-to-create-a-directory-in-c%2b%2b/
- Microsoft. (n.d.). <filesystem> functions. Microsoft | Docs. <https://docs.microsoft.com/en-us/cpp/standard-library/filesystemfunctions?view=msvc-170/>
- Deitel, P., & Deitel, H. (2014). C++ How to program, (9th ed.). Pearson.
- Laudo, J. et al. (2022). Library management system. GitHub. <https://github.com/InLurker/Library-Management-System-Cpp/>
- DelftStack. (2020). Get current directory in C++. <https://www.delftstack.com/howto/cpp/get-current-directory-cpp/>
- Shaffer, C. (2011). A practical introduction to data structures and algorithm analysis, 47 (3.2 ed.). Virginia Tech.DevDocs. (n.d.). Standard library header <filesystem>. <https://devdocs.io/cpp/header/filesystem/>
- Ng, K.W., Ma, J. and Nam, G.M., 1993, March. A class library management system for object-oriented programming. In Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice (pp. 445-451).
- Hameed, J.M., 2012. Design and Implementation of Probabilistic Assignment Management System for a Library Database Using Object Oriented Programming. AL-Rafidain Journal of Computer Sciences and Mathematics, 9(2), pp.99-108.
- DelftStack. (2021). Create directory in C++. https://www.delftstack.com/howto/cpp/cpp-create-directory/#use-the-stdfilesystemcreate_directories-function-to-create-a-directory-in-c%2b%2b/

Appendix

Book	User
<ul style="list-style-type: none"> → Title: String → ISBN: String → Status: String → Author: String → Available Copies: String → Reserved Id: String → Display () : Void → Get & Set Methods <ul style="list-style-type: none"> ○ getTitle(): String ○ getAuthor(): String ○ getISBN(): String ○ getType(): String ○ getAvailableCopies(): Int ○ getReservedIds():vector<int> ○ setTitle: Void ○ setAuthor: Void ○ setISBN: Void ○ setType: Void ○ setAvailableCopies: Void ○ setReservedIds: Void ○ addReservedId: Void 	<ul style="list-style-type: none"> → Id: Int → Name: String → Type: String → Password: String → Get & Set Methods <ul style="list-style-type: none"> ○ getId(): Int ○ getName(): String ○ getType(): String ○ getPassword(): String ○ setId(): Void ○ setName(): Void ○ setType(): Void ○ setPassword(): Void ○ display(): Void
Library	Borrower
<ul style="list-style-type: none"> → Book: vector<Book> → Borrower: vector<Borrower> → User: vector<User> → loadBooks: Void → addUser(): Void → saveUsers: Void → insertBorrower(): Void → searchBorrowerById: Void → checkUserLogin: Void → returnBook: Void → checkUserPresence: Bool 	<ul style="list-style-type: none"> → Borrowed Books: vector<string> → Get & Set Methods <ul style="list-style-type: none"> ○ borrowBook: Void ○ setBorrowedBooks: Void ○ returnBook: Void ○ hasBorrowedBook: Bool ○ addBorrowedBook: Void ○ removeBorrowedBook: Void

Figure 14: Application Index

Individual Report

• Walid KH.J Suleiman -202169990015

Introduction:

In this project, I was assigned the task of completing the implementation of the Book class for a library management system. My goal was to ensure that the Book class accurately represented a book and included the necessary attributes and methods. Additionally, I

implemented the main function, which serves as the entry point of the program and orchestrates the overall flow of the library management system.

Book Class Implementation:

To begin with, I defined the Book class using the "class" keyword, which allowed me to create book objects based on this blueprint. Inside the class, I added several attributes to represent different aspects of a book:

Title: This attribute stores the title of the book, enabling each book object to hold its specific title information.

Author: I included the author attribute to store the name of the book's author. This attribute plays a crucial role in identifying and retrieving book information accurately.

ISBN: The ISBN attribute provides a standardized identifier for each book. This unique identifier allows for efficient cataloging and referencing within the library system.

Borrowed: To track the borrow status of each book, I added the borrowed attribute, which serves as a boolean flag. It indicates whether the book is currently borrowed or available for borrowing.

BorrowerID: This attribute stores the ID of the borrower who currently has the book on loan. When a book is available, the borrowerID is set to -1, indicating that no borrower is associated with the book. However, when a borrower checks out the book, the borrowerID is updated with the respective borrower's ID, establishing the link between the book and the borrower.

To enhance the functionality of the Book class, I implemented getter and setter methods. The getter methods (getTitle(), getAuthor(), getISBN()) allow users to retrieve the book's title, author, and ISBN, respectively. These methods provide convenient ways to access the book's information for display or further processing. On the other hand, the setter methods (setBorrowed(), setBorrowerID()) enable users to modify the book's properties. By invoking these methods, users can update the book's borrow status and set the borrower ID, facilitating the borrowing process.

Challenges and Difficulties:

During the implementation of the Book class, I encountered a few challenges and difficulties. Here are some notable ones:

Understanding the requirements: Initially, I needed to thoroughly understand the requirements and specifications of the Book class. This involved analyzing the desired attributes, their relationships, and the expected behavior of the class methods. Clear understanding of the requirements was essential to ensure accurate implementation.

Designing an efficient identifier: The selection and implementation of a unique identifier for each book was a critical decision. I needed to consider the constraints and standards associated with book identifiers, such as the ISBN. Designing an efficient identifier system required careful consideration and adherence to industry standards.

Managing borrow status and borrower association: Implementing the borrow status and borrower association required careful handling of data and maintaining the integrity of the

book and borrower objects. Ensuring the appropriate updates to the borrow status and borrower ID while handling edge cases, such as multiple borrowers or unassociated books, required thoughtful consideration and rigorous testing.

Ensuring data encapsulation: While implementing the getter and setter methods, I had to ensure proper encapsulation of the book's properties. This involved setting access modifiers and designing the methods to provide controlled access to the attributes, preventing direct modification and ensuring data consistency.

What I Have Learned:

Throughout this project, I gained valuable experience and learned several important concepts and skills, including:

Object-oriented programming (OOP): Implementing the Book class allowed me to practice the principles of OOP. I learned how to design and define classes, encapsulate data and behavior, and create instances of objects based on those classes.

Class attributes and methods: I learned how to identify and define the attributes and methods required to accurately represent a book object. Understanding the relationship between attributes and their role in encapsulating data was essential for building a well-structured class.

Data validation and integrity: I encountered the importance of implementing data validation mechanisms to ensure the accuracy and integrity of the book data. Validating attributes like title, author, and ISBN to meet the required criteria was crucial for maintaining the reliability of the library system.

Handling user input and user interfaces: Implementing the main function allowed me to practice handling user input and designing menu-based interfaces. I learned how to prompt users for input, process their choices, and present appropriate messages or options based on their selection.

File I/O operations: Loading initial data from files and saving updated data back to files provided valuable experience with file input/output operations. I learned how to read and write data to files, enabling the persistence and retrieval of library system information.

Special Challenges:

Several challenges posed unique considerations during the implementation of the Book class and the main function:

Designing an effective and user-friendly interface: Developing a user interface that accommodates both administrators and regular users required careful consideration of their distinct needs. Ensuring an intuitive and logical flow of options and providing informative error messages added complexity to the implementation.

Authentication and access control: Implementing authentication mechanisms to differentiate between administrators and regular users and managing user accounts introduced challenges. Ensuring password validation, maintaining user records, and enforcing access control required thoughtful design and implementation.

Data consistency and error handling: Ensuring data consistency while performing operations like book borrowing, returning, or user account management was crucial. Handling various scenarios, such as incorrect user input, edge cases, or unexpected errors, required robust error handling strategies and thorough testing.

Conclusion:

In conclusion, the completion of the Book class implementation and the main function for the library management system involved overcoming challenges, acquiring new skills, and applying key learnings. The implementation of the Book class allowed for accurate representation of book objects, incorporating essential attributes and methods. The main function facilitated the control flow and interaction logic of the library management system, providing users and administrators with a menu-based interface for performing various operations on books, borrowers, and library data. The difficulties faced, key learnings, and special challenges encountered throughout the project demonstrate the breadth of knowledge and skills acquired in class design, data handling, user interaction, and system functionality.

• Ibrahim Alfaqih-20216990079

Introduction:

In this project, my assigned task was to complete the implementation of the User class for a library management system. The User class is responsible for representing users of the library and includes attributes and methods necessary to accurately represent a library user. I also added the necessary connections and dependencies between the User class and other components of the library management system.

User Class Implementation:

To start, I defined the User class using the "class" keyword, which allowed me to create user objects based on this blueprint. Inside the class, I added several attributes to represent different aspects of a user:

1. **userID:** This attribute stores the unique identifier of each user. It enables the library system to differentiate between different users and associate their actions and records correctly. Each user object holds its specific userID information, facilitating efficient tracking and management of user-related activities.
2. **userType:** I included the userType attribute to categorize users into different types, such as students, teachers, or researchers. This attribute allows for differentiation of user privileges, access levels, or specific functionalities based on the user's role within the library system. The userType attribute enables personalized interactions and customization for different user categories.
3. **password:** To address user authentication and security, I added the password attribute. This attribute stores a password associated with each user, ensuring secure access to user-specific functionalities and data. The password attribute prevents unauthorized access and protects sensitive user information within the library system.
4. **userName:** The userName attribute represents the name of the user. It allows each user object to store the individual's name, facilitating personalized interactions and identification.

within the library system. The `userName` attribute enhances the user experience by providing a human-readable identifier for users.

To provide convenient access to the user's attributes, I implemented getter methods such as `getUserID()`, `getUserType()`, `getUserName()`, and `getPassword()`. These methods allow users of the `User` class to retrieve the user's ID, type, name, and password, respectively. The getter methods provide a way to access user information for authentication, display, or further processing. To ensure proper authentication and access control, I implemented a `checkUserPresence()` method. This method takes the `userID`, `userType`, and `password` as parameters and verifies whether a user with the provided credentials exists in the library system. The `checkUserPresence()` method plays a crucial role in validating user authentication and enabling appropriate access to the system's functionalities based on user privileges.

Difficulties and Challenges:

During the implementation of the `User` class, I encountered several challenges and difficulties. Here are some notable ones:

1. **Authentication and Security:** Designing a robust authentication system that ensures secure user access and protects sensitive information required careful consideration. Implementing mechanisms to securely store and validate user passwords presented challenges related to encryption, hashing, or password policies.
2. **User Roles and Privileges:** Defining user types and managing their associated roles and privileges required thoughtful design and implementation. Ensuring appropriate access levels and functionalities based on user types involved considering the specific requirements of different user categories, such as students, teachers, or administrators.
3. **User Management and Data Consistency:** Implementing user management functionalities, such as creating new user accounts, updating user information, or deleting user accounts, posed challenges related to data consistency and maintaining integrity across the library management system. Ensuring data consistency while handling user-related operations required careful handling of edge cases and robust error handling.
4. **User Interface Integration:** Integrating the `User` class with the user interface and other components of the library management system involved challenges related to data flow, parameter passing, and synchronization. Ensuring seamless communication and interaction between the `User` class and the user interface required thorough testing and coordination.

What I Have Learned:

Throughout this project, I gained valuable experience and learned several important concepts and skills, including:

1. **User Representation:** I learned how to accurately represent users within a system by identifying and defining the necessary attributes and methods. Understanding the significance of attributes like `userID`, `userType`, `password`, and `userName` in representing users was crucial for building a comprehensive `User` class.

2. Authentication and Access Control: Implementing user authentication mechanisms and access control logic provided insights into secure user management. Learning about password management, authentication protocols, and user role-based access control enhanced my understanding of security-related aspects of software systems.

3. Object-Oriented Programming: The implementation of the User class allowed me to reinforce my understanding of object-oriented programming principles. Designing the class structure, encapsulating attributes and methods, and managing class dependencies contributed to my proficiency in building object-oriented systems.

4. Data Validation and Integrity: Ensuring data consistency, validating user inputs, and enforcing business rules within the User class improved my understanding of data validation and integrity principles. Implementing validation mechanisms for attributes like passwords or user types helped me grasp the importance of data validation in maintaining system reliability.

Conclusion:

In conclusion, completing the implementation of the User class in the library management system involved addressing challenges related to authentication, user roles, data consistency, and user interface integration. The User class incorporates attributes like `userID`, `userType`, `password`, and `userName` to accurately represent users and manage their interactions within the library system. Getter methods provide access to user attributes, while the `checkUserPresence()` method ensures proper authentication and access control. Through this project, I deepened my understanding of user representation, authentication mechanisms, object-oriented programming, and data validation principles.

- **Subodh Pokhrel-202169990044**

Introduction:

In this project, my assigned task was to complete the implementation of the Borrower class for a library management system. The Borrower class represents individuals who borrow books from the library and includes attributes and methods necessary to accurately represent a borrower. I also added the necessary connections and dependencies between the Borrower class and other components of the library management system.

Borrower Class Implementation:

To begin, I defined the Borrower class using the "class" keyword, which allowed me to create borrower objects based on this blueprint. Inside the class, I added several attributes to represent different aspects of a borrower:

1. `borrowerID`: This attribute serves as a unique identifier for each borrower. It allows the library system to differentiate between different borrowers and maintain accurate records of their borrowing activities. Each borrower object stores its specific `borrowerID` information, enabling efficient tracking and management of borrower-related actions.

2. `borrowedBooks`: To keep track of the books borrowed by a borrower, I included the `borrowedBooks` attribute. This attribute is implemented as a container, such as a list or an array, to store the books borrowed by a particular borrower. It allows the library system to maintain a record of the books currently in the possession of each borrower.

3. **reservedBooks**: In addition to borrowed books, I recognized the need to keep track of reserved books. For this purpose, I introduced the **reservedBooks** attribute. Similar to **borrowedBooks**, **reservedBooks** is implemented as a container to store the books reserved by the borrower. This attribute enables the library system to manage book reservations and ensure that reserved books are available when requested. To provide convenient access to the borrower's attributes, I implemented getter methods such as **getBorrowerID()**, **getBorrowedBooks()**, and **getReservedBooks()**. These methods allow users of the **Borrower** class to retrieve the borrower's ID, the list of borrowed books, and the list of reserved books, respectively. The getter methods facilitate access to borrower information for display, further processing, or generating reports. To enable borrowers to borrow books, I implemented the **borrowBook()** method. This method takes a book object as a parameter and adds it to the borrower's **borrowedBooks** container. The **borrowBook()** method is responsible for updating the borrower's borrowing records and ensuring that the borrowed book is associated with the borrower. Similarly, I implemented the **reserveBook()** method, which allows borrowers to reserve books. The **reserveBook()** method takes a book object as a parameter and adds it to the borrower's **reservedBooks** container. This method ensures that the borrower's reservation is recorded and managed appropriately by the library system.

Difficulties and Challenges:

During the implementation of the **Borrower** class, I encountered several challenges and difficulties. Here are some notable ones:

1. **Data Management and Consistency**: Managing borrowed and reserved books for each borrower while ensuring data consistency required careful consideration. Implementing proper data structures and synchronization mechanisms to handle simultaneous borrowing and reservation requests was challenging.
2. **Handling Book Availability**: Ensuring that reserved books are available when requested by borrowers involved challenges related to book availability management. Coordinating book reservations with the overall book inventory and handling cases where reserved books are already borrowed required careful logic implementation.
3. **Integration with Book and Library Classes**: Connecting the **Borrower** class with the **Book** class and the library management system as a whole involved challenges related to data flow, dependency management, and ensuring consistency across different components. Proper integration was crucial for accurate book tracking and borrower management.
4. **Error Handling and Exception Management**: Implementing robust error handling and exception management mechanisms to handle scenarios like invalid book reservations, borrowing limits, or book availability issues required thorough testing and consideration of various edge cases.

What I Have Learned:

Throughout this project, I gained valuable experience and learned several important concepts and skills, including:

1. Data Modeling: Design

ing the Borrower class allowed me to practice data modeling and representation, specifically in the context of borrowers and their interactions with the library system. Identifying relevant attributes and relationships contributed to my understanding of data modeling principles.

2. Object-Oriented Design: Implementing the Borrower class enhanced my knowledge of object-oriented design principles. Building the class structure, encapsulating attributes and methods, and managing class interactions reinforced my understanding of object-oriented programming concepts.

3. Dependency Management: Establishing connections and dependencies between the Borrower class, Book class, and the library management system demonstrated the importance of proper dependency management. Understanding how classes and components interacted with each other helped me grasp the significance of cohesive and loosely coupled systems.

4. Error Handling and Exception Management: Dealing with potential errors and exceptions, such as invalid book reservations or borrowing limits, required me to implement effective error handling mechanisms. This experience improved my ability to anticipate and handle exceptional situations in software development.

Conclusion:

In conclusion, completing the Borrower class implementation in the library management system involved addressing challenges related to data management, book availability, integration, and error handling. The Borrower class incorporates attributes like `borrowerID`, `borrowedBooks`, and `reservedBooks` to accurately represent borrowers and manage their borrowing and reservation activities. Getter methods facilitate access to borrower attributes, while the `borrowBook()` and `reserveBook()` methods enable borrowers to perform borrowing and reservation actions seamlessly. This project provided valuable insights into data modeling, object-oriented design, dependency management, and error handling.

• Sudipta Sotra Dhar-20216990264

Introduction:

The Library class is a crucial component of the library management system, responsible for managing books, borrowers, and various library operations. In this project, my task was to complete the implementation of the Library class, ensuring it contains the necessary attributes and methods to effectively handle library functionalities. This report highlights the completed implementation, including the challenges faced and the lessons learned during the process.

Library Class Implementation:

To begin, I defined the Library class using the "class" keyword, establishing the structure for creating library objects and encapsulating related operations within this class. Inside the Library class, I included several attributes to store and manage books, borrowers, and other relevant data:

1. books: This attribute represents the collection of books in the library. Implemented as a container, such as a list or an array, it allows for the storage of multiple book objects. The

books attribute enables efficient maintenance and manipulation of the book inventory within the library system.

2. **borrowers**: This attribute serves as a container for storing borrower objects, facilitating the tracking of library patrons, their borrowing history, and other borrower-related information. It allows the library system to manage borrower records and track their interactions with the library.

3. **users**: The users attribute stores user accounts for the library system, enabling authentication and access control. This attribute provides the necessary means to manage different user roles, such as administrators and regular users, ensuring appropriate access to library functionalities based on user credentials.

To support various library operations, I implemented methods within the Library class. These methods provide the necessary functionality to interact with the book and borrower data, perform searches, insert new entries, modify existing records, and delete items as required. Some notable methods include:

1. **insertBook()**: This method allows administrators to add new books to the library. It takes book details as parameters, creates a new book object, and adds it to the books container.

2. **modifyBook()**: The `modifyBook()` method enables administrators to update the information of an existing book. It takes the ISBN of the book to be modified as a parameter, searches for the book in the books container, and updates the relevant attributes with the provided information.

3. **deleteBook()**: This method facilitates the removal of a book from the library. It takes the ISBN of the book to be deleted, searches for it in the books container, and removes it from the collection, effectively eliminating the book from the library inventory.

4. **insertBorrower()** and **modifyBorrower()**: These methods handle borrower-related operations, allowing administrators to add new borrowers to the library system or modify the information of existing borrowers. They interact with the borrowers container, updating the relevant borrower attributes accordingly.

5. **borrowBook()**, **reserveBook()**, and **returnBook()**: These methods support book borrowing and reservation functionalities. They interact with the book and borrower data, updating the relevant attributes to facilitate the borrowing, reservation, and return processes.

In addition to these methods, I implemented book search methods that allow users to locate specific books within the library based on different criteria such as title, author, or ISBN.

Difficulties and Challenges:

During the implementation of the Library class, I encountered several challenges and difficulties:

1. **Data Organization and Management**: Managing and organizing book and borrower data efficiently required careful consideration of data structures and algorithms. Ensuring data consistency and handling concurrent operations posed challenges that required thoughtful implementation.

2. Authentication and Access Control: Implementing user authentication and access control mechanisms required a robust approach. Managing user accounts, roles, and access privileges while ensuring data security and integrity involved careful design and implementation.

3. Integration with Other Classes: Connecting the Library class with other classes, such as Book and Borrower, and ensuring smooth interactions and data flow presented challenges related to dependency management and overall system design. Proper integration and coordination were crucial for accurate book tracking, borrowing, and reservation processes.

4. Error Handling and Validation: Implementing error handling mechanisms, validating user inputs, and handling exceptional situations, such as invalid book reservations or exceeding borrowing limits, required careful consideration. Ensuring proper error messages and recovery strategies enhanced the usability and reliability of the library management system.

What I Have Learned:

Completing the implementation of the Library class provided valuable lessons and insights:

1. Data Modeling: Designing and implementing the Library class involved data modeling and representation, emphasizing the importance of identifying relevant attributes and relationships. Understanding how to model real-world entities into software objects improved my data modeling skills.

2. Object-Oriented Design: Implementing the Library class deepened my understanding of object-oriented design principles. Structuring the class, encapsulating attributes and methods, and managing interactions between classes enhanced my knowledge of object-oriented programming concepts.

3. System Integration: Connecting the Library class with other classes highlighted the significance of proper system integration. Understanding how different components interacted and coordinating their functionalities reinforced the importance of cohesive and modular system design.

4. Usability and Error Handling: Implementing user-friendly interfaces, proper error handling, and validation mechanisms improved the usability and reliability of the library management system. Understanding the importance of user input validation and effective error messages enhanced the overall user experience.

Conclusion:

In conclusion, completing the implementation of the Library class in the library management system involved overcoming challenges related to data organization, authentication, integration, and error handling. The Library class incorporates attributes such as books, borrowers, and users to store and manage relevant data. The implemented methods enable administrators and users to perform various library tasks, such as adding, modifying, and deleting books, managing borrower information, and facilitating book borrowing and reservation. This project provided valuable insights into data modeling, object-oriented design, system integration, and usability considerations.