ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG QUỐC TẾ
VNU-INTERNATIONAL SCHOOL

Course title: PROGRAMMING 2

FINAL EXAM ASSIGNMENT

Topic: VNU IS's Library Database Management System

Lecturer : Nguyễn Đình Trần Long

Group member : Đào Thị Phương Anh - 22070895
Nguyễn Thái Việt Anh - 22070884
Nguyễn Quỳnh Chi - 22071081
Nguyễn Thị Kim Oanh - 22070937
Nguyễn Đức Hiếu - 21070553

Class : INS207301

Hanoi, May 2024

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION

The Library Management System is a comprehensive software solution designed to streamline the operations and management of a library. This system aims to provide an efficient and user-friendly platform for librarians, staff, and patrons to effectively manage the library's collection, borrowing activities, and patron information.

- ***About the Organization***

The Library Management System is developed for a local public library that serves a diverse community. The library houses a collection of over 50,000 physical books, as well as a growing selection of e-books and other digital resources. With a goal of promoting literacy, education, and knowledge-sharing, the library seeks to provide its patrons with easy access to a wide range of materials and resources.

- ***Objectives***

The primary objectives of the Library Management System are:

1. Efficient Book Management: Provide a centralized system to catalog, organize, and maintain the library's collection of books and other materials.
2. Patron Management: Facilitate the registration, authentication, and management of library patrons, including their borrowing history and account details.
3. Loan and Return Tracking: Enable the seamless tracking of book loans, returns, and reservations to ensure the effective utilization of the library's resources.
4. Reporting and Analytics: Generate comprehensive reports and analytics to assist librarians in making informed decisions about acquisitions, resource allocation, and patron engagement.
5. User-Friendly Interface: Develop a intuitive and accessible user interface that caters to the needs of librarians, staff, and patrons, ensuring a smooth and enjoyable user experience.

- ***Scope***

The scope of the Library Management System includes the following key features and functionalities:

1. Book Management:
   - Add, edit, and remove book records, including details such as title, author, publication date, and genre.
   - Manage the availability status of books (e.g., available, loaned, reserved).
   - Maintain a comprehensive catalog of the library's book collection.

2. Author Management:
   - Add, edit, and remove author records, including their biographical information and the books they have authored.
   - Establish relationships between authors and their published books.

3. Patron Management:
   - Register new patrons and maintain their personal and contact information.
   - Manage patron accounts, including borrowing privileges, fines, and login credentials.
   - Track the borrowing history and current loans for each patron.

4. Loan and Return Management:
   - Check out books to patrons and record the loan details, including the due date.
   - Facilitate the return of borrowed books and update the availability status accordingly.
   - Manage book reservations and holds placed by patrons.

5. Searching and Reporting:
   - Allow users to search for books by title, author, or patron details.
   - Generate reports on the library's collection, loan statistics, and patron activities.
   - Provide analytics and insights to support decision-making and resource management.

By implementing the Library Management System, the library aims to enhance the user experience for both librarians and patrons, improve the efficiency of its operations, and ultimately, foster a thriving and well-informed community.

CHAPTER 2: SYSTEM OVERVIEW

The Library Management System is designed to manage a library's operations including books, authors, patrons, and loans. It provides functionalities for adding, removing, searching, and updating details for each entity. The system ensures data persistence by saving and loading data from text files. Here's an overview of the system's structure and functionality:

- ***Class Overview***

1. Book:
   - Attributes: title, author, year
   - Methods: display()

2. Author:
   - Attributes: authorID, name, books (a list of pointers to Book objects)
   - Methods: addBook(Book* book), display()

3. Patron:
   - Attributes: patronID, name
   - Methods: display()

4. Loan:
   - Attributes: loanID, book (pointer to Book), patron (pointer to Patron), loanDate, returnDate
   - Methods: display()

5. Library:
   - Attributes:
     + books (a vector of Book objects)
     + authors (a vector of Author objects)
     + patrons (a vector of Patron objects)
     + loans (a vector of Loan objects)
     + File names for storing data: bookFile, authorFile, patronFile, loanFile
   - Methods:
     + Data Loading and Saving: loadBooks(), loadAuthors(), loadPatrons(), loadLoans(), saveBooks(), saveAuthors(), savePatrons(), saveLoans()
     + Add Entities: addBook(const Book&), addAuthor(const Author&), addPatron(const Patron&), addLoan(const Loan&)
     + Remove Entities: removeBook(const string&), removeAuthor(const string&), removePatron(int), removeLoan(int)

+ Display Entities: displayBooks(), displayAuthors(), displayPatrons(), displayLoans()

+ Search Entities: searchBookByTitle(const string&), searchAuthorByName(const string&), searchPatronByID(int)

+ Update Entities: updateBookDetails(const string&), updateAuthorDetails(const string&, const string&), updatePatronDetails(int, const string&)

+ Getters for Testing and Integration: getBooks(), getAuthors(), getPatrons(), getLoans()

- ***Main Program***

- User Interface: Provides a command-line interface for interacting with the system.
- Menu Options:
  1. Add Book
  2. Remove Book
  3. Add Author
  4. Remove Author
  5. Add Patron
  6. Remove Patron
  7. Add Loan
  8. Remove Loan
  9. Display All Data
  10. Search Book by Title
  11. Search Author by Name
  12. Search Patron by ID
  13. Update Book Details
  14. Update Author Details
  15. Update Patron Details
  16. Exit

- ***Detailed Workflow***

1. Adding Entities: Users input data for books, authors, patrons, and loans. The system creates objects and adds them to the respective vectors, then saves the updated data to the corresponding files.

2. Removing Entities: Users specify the identifier (e.g., title for books, ID for authors/patrons/loans). The system searches for the entity, removes it from the vector, and updates the file.

3. Displaying Data: Lists all entries in each category by iterating through the vectors and calling the `display()` method on each object.

4. Searching: Users input search criteria (e.g., book title, author name, patron ID). The system searches the vector for a match and displays the corresponding entity if found.

5. Updating Details: Users specify the identifier of the entity to update and provide new details. The system updates the object's attributes and saves the updated vector to the file.

6. Persistence: Upon initialization, the system loads data from files. Upon termination, it saves all data back to the files, ensuring persistence across sessions.

This structure ensures a modular, maintainable, and extendable system for managing a library's data effectively.

CHAPTER 3: PROGRAM DESIGN

- **Class Book** : class Book {
  Overview: "The Book class is designed to represent a book in a library management system. This class includes attributes for the title, author, and year of publication of the book."

1. Public attributes:

string title;	*// This is a public attribute of the `Book` class, used to store the title of the book.*
string author;	*// This is a public attribute of the `Book` class, used to store the author's name of the book.*
int year;	*// This is a public attribute of the `Book` class, used to store the publication year of the book.*

2. Constructor:

Book(string t, string a, int y) : title(t), author(a), year(y) {}	*// This is the constructor of the `Book` class. It takes three parameters: `t` (title), `a` (author), and `y` (year). In the body of the constructor, the public attributes `title`, `author`, and `year` are initialized by assigning the values of the corresponding parameters.*

3. Display() method:

void display() const	*// This is a public method of the `Book` class named display() . It does not take any parameters and does not return any value.*

- **Class Author**: class Author{
  Overview: "The Author class is designed to represent an author in a library management system. This class includes attributes for the author's ID and name, as well as a collection of books written by the author."

1.Public Attributes:

```
string authorID;
```
// This is a public attribute that stores the unique identifier for the author.

```
string name;
```
// This is a public attribute that stores the name of the author.

```
vector<Book*> books
```
// This is a public attribute that stores a vector of pointers to Book objects, representing the books written by the author.

2. Constructor:

```
Author(string id, string n) : authorID(id), name(n) {}
```
// This is the constructor for the Author class. It takes two parameters, id (the author's ID) and n (the author's name), and initializes the authorID and name attributes with the corresponding values.

3. AddBook() method:

```
void addBook(Book* book)
```
// This is a public method that allows adding a new Book object (referenced by a pointer) to the books vector of the Author object.

```
books.push_back(book)
```
// The Book pointer is added to the books vector.

4. Display() method:

```
void display()
```
// This is a public method that displays the information about the Author object.

```
for (const auto& book : books)
```
// A loop is used to iterate through the books vector.

```
book->display()
```
// For each Book object in the books vector, the display() method of the Book object is called to print the book's information.

- **Class Patron** : class Patron{
  Overview: “The Patron class is used to represent a library patron, which is a person who borrows books from the library.”

1. Attributes.

int patronID;	*// This is an integer that uniquely identifies each patron. It serves as a unique ID for the patron in the library system.*
string name;	*// This is a string that stores the name of the patron.*

2. Constructor.

Patron(int id, string n)	*// This constructor initializes a `Patron` object with an ID (`id`) and a name (`n`). When a new `Patron` object is created, this constructor is called to set the initial values for `patronID` and `name`.*
patronID(id), name(n)	*// This is an initialization list used to initialize the member variables `patronID` and `name` with the values `id` and `n` respectively.*

3. Display() Method.

void display() const	*// This method is used to display the details of a patron. The `const` keyword indicates that this method does not modify the object's state.*
cout << "Patron ID: " << patronID << ", Name: " << name << endl	*// This line outputs the patron's ID and name to the console. The `cout` object is used for standard output in C++.*

- **Class Loan** : class Loan{
  Overview: “The Loan class is designed to represent a loan record in a library management system. This record links a book to a patron, including details about the loan and return dates.”

1. Attributes.

`int loanID;`	*// This is an integer that uniquely identifies each loan. It serves as a unique ID for the loan record in the library system.*
`Book* book;`	*// This is a pointer to a `Book` object, representing the book that is loaned out.*
`Patron* patron;`	*// This is a pointer to a `Patron` object, representing the patron who borrowed the book.*
`string loanDate;`	*// This is a string that stores the date when the book was loaned.*
`string returnDate;`	*// This is a string that stores the date when the book is expected to be returned or was returned.*

2. Constructor

`Loan(int id, Book* b, Patron* p, string lDate, string rDate)`	*// This constructor initializes a `Loan` object with an ID (`id`), a pointer to a book (`b`), a pointer to a patron (`p`), a loan date (`lDate`), and a return date (`rDate`).*
`loanID(id), book(b), patron(p), loanDate(lDate), returnDate(rDate) {}`	*// This is an initialization list used to initialize the member variables `loanID`, `book`, `patron`, `loanDate`, and `returnDate` with the provided values.*

3. display() Method.

`void display() const`	*// This method is used to display the details of a loan record. The `const` keyword indicates that this method does not modify the object's state.*

```
cout << "Loan ID: " << loanID << ", Book Title: " << book->title << ", Patron ID: " << patron->patronID << ", Loan Date: " << loanDate << ", Return Date: " << returnDate << endl
```

// This line outputs the loan's ID, the title of the loaned book, the ID of the patron who borrowed the book, the loan date, and the return date to the console. The `cout` object is used for standard output in C++.

- **Class Library** : class Library {
  Overview: "The Library class manages books, authors, patrons, and loans. It now includes methods for loading data from and saving data to text files, ensuring that the library's state is persistent between program runs."

  Role: "Manages collections of books, authors, patrons, and loans, and handles loading from and saving to files."
  Private Data Members

1.Member Variable

```
vector<Book> books;
vector<Author> authors;
vector<Patron> patrons;
vector<Loan> loans;
```

// books: This vector stores information about books in the library. (likely Book objects)
// authors: This vector stores information about authors. (likely Author objects)
// patrons: This vector stores information about library patrons. (likely Patron objects)
// loans: This vector stores information about book loans. (likely Loan objects).

2. File part

```
const string bookFile = "books.txt";
const string authorFile = "authors.txt";
const string patronFile = "patrons.txt";
const string loanFile = "loans.txt";
```

// bookFile: Path the to file containing book information.
// authorFile: Path to the file containing author information
// patronFile: Path to the file containing patron information
// loanFile: Path to the file containing loan information.

3. Loading Data Functions:

```
void loadBooks() {
      ifstream file(bookFile);
      if (file.is_open()) {
         string title, author;
         int year;
         while (file >> ws &&
getline(file, title) && getline(file,
author) && file >> year) {
            books.emplace_back(title,
author, year);
            file.ignore();  // Ignore
newline after the year
         }
         file.close();
      }
   }
```

// loadBooks(): This function reads book information from the bookFile.

- *using "is_open()" to check the file was opened successfully*
- *Inside a loop, it reads the title, author, and year for each book.*
- *It creates a Book object (assuming Book class exists) with the read data and adds it to the books vector.*
- *file.ignore(); :This ignore() call discards any remaining characters (including the newline) on the current line before moving to the next line in the file. Otherwise, the next getline would attempt to read an empty line.*

```
void loadAuthors() {
      ifstream file(authorFile);
      if (file.is_open()) {
         string authorID, name;
         while (file >> ws &&
getline(file, authorID) &&
getline(file, name)) {

authors.emplace_back(authorID,
name);
            file.ignore();
         }
         file.close();
      }
   }
```

// loadAuthors(): This function follows a similar structure as loadBooks but reads author data from authorFile and creates Author objects (assuming Author class exists), storing them in the authors vector.

```
   void loadPatrons() {
      ifstream file(patronFile);
      if (file.is_open()) {
         string name;
         int patronID;
         while (file >> ws &&
getline(file, name) && file >>
patronID) {
```

// loadPatron(): Similar to the previous functions, it reads patron daa from patronFile, creates Patron objects, and stores them in the patrons vector.

```
patrons.emplace_back(patronID,
name);
            file.ignore();
        }
        file.close();
    }
}
```

```
void loadLoans() {
    ifstream file(loanFile);
    if (file.is_open()) {
        int loanID;
        string loanDate, returnDate;
        while (file >> ws && file >>
loanID && getline(file, loanDate)
&& getline(file, returnDate)) {
            // Placeholder book and
patron, you need to find the correct
ones in main

loans.emplace_back(loanID, nullptr,
nullptr, loanDate, returnDate);
            file.ignore();
        }
        file.close();
    }
}
```

// loadLoans(): This function reads loan information from loanFile.

- *It follows a similar loop structure but encounters a placeholder for book and patron objects.*
- *It reads loan ID, loan date, and return date for each loan.*
- *It creates a Loan object (assuming Loan class exists) with the data but uses temporary nullptr values for book and patron.*
- *These placeholders will likely be filled in the main program by finding the corresponding book and patron objects based on IDs.*

4. Saving Data Functions: writing the book information stored in the library system back to a file for persistence.

```
void saveBooks() const {
    ofstream file(bookFile);
    if (file.is_open()) {
        for (const auto& book : books)
{
            file << book.title << endl;
            file << book.author << endl;
            file << book.year << endl;
        }
```

// Opening the file

// Checking for success

```
        file.close();
    }
}

void saveAuthors() const {
    ofstream file(authorFile);
    if (file.is_open()) {
        for (const auto& author :
authors) {
            file << author.authorID <<
endl;
            file << author.name <<
endl;
        }
        file.close();
    }
}

void savePatrons() const {
    ofstream file(patronFile);
    if (file.is_open()) {
        for (const auto& patron :
patrons) {
            file << patron.patronID <<
endl;
            file << patron.name <<
endl;
        }
        file.close();
    }
}

void saveLoans() const {
    ofstream file(loanFile);
    if (file.is_open()) {
        for (const auto& loan : loans)
{
            file << loan.loanID << endl;
            file << loan.book->title <<
endl;
            file <<
loan.patron->patronID << endl;
            file << loan.loanDate <<
endl;
            file << loan.returnDate <<
```

// saveAuthors(): Similar to saveBooks(), it writes author information back to the authorFile.

// savePatrons(): Similar to the above functions, it writes patron information back to the patronFile.

// saveLoans(): This function writes loan information back to the loanFile. Unlike others, it uses the actual book title retrieved from the book object (assuming a getter method exists) and patron ID retrieved from the patron object (assuming a getter method exists).

```
endl;
            }
            file.close();
        }
    }
```

5. Constructor (Library()):

```
public:
    Library() {
        loadBooks();
        loadAuthors();
        loadPatrons();
        loadLoans();
    }
```

- *This constructor is called whenever a new Library object is created.*
- *Its primary responsibility is to initialize the library by loading data from persistent storage.*
- *The code includes the following data loading functions:*
  - *loadBooks(): Loads book information (title, author, year) from the books.txt file.*
  - *loadAuthors(): Loads author information (author ID, name) from the authors.txt file.*
  - *loadPatrons(): Loads patron information (patron ID, name) from the patrons.txt file.*
  - *loadLoans(): Loads loan information (loan ID, book title [placeholder], patron ID [placeholder], loan date,return date) from the loans.txt file.*

6. Destructor (~Library()):

```
~Library() {

        saveBooks();
        saveAuthors();
        savePatrons();
        saveLoans();
    }
```

- *This destructor is called automatically when a Library object goes out of scope or is explicitly deleted.*
- *Its main purpose is to ensure all data stored in memory is saved back to the corresponding text files before the object is destroyed.*
- *The code calls the following data saving functions:*
  - *saveBooks(): Saves book information from memory to the books.txt file.*
  - *saveAuthors(): Saves author information from memory to the authors.txt*

file.

- *savePatrons(): Saves patron information from memory to the patrons.txt file.*
- *saveLoans(): Saves loan information from memory to the loans.txt file.*

6.1 Add methods

```
void addBook(const Book& book) {
    books.push_back(book);
    saveBooks();
}

void addAuthor(const Author&
author) {
    authors.push_back(author);
    saveAuthors();
}

void addPatron(const Patron&
patron) {
    patrons.push_back(patron);
    savePatrons();
}

void addLoan(const Loan& loan) {
    loans.push_back(loan);
    saveLoans();
```

// This function adds a new book (book) to the library's collection.

// This is the parameter of the function. It takes a constant reference to an Author object. This means the function can access the data of the author object but cannot modify it.

// Adding patron

// Saving patron

// Adding loan and Save

```
void addBook(const Book& book) {
    books.push_back(book);
    saveBooks();
}
```

// Adding book. (const Book& book) the function can access the information of the Book object but cannot modify the original object itself.

```
void addAuthor(const Author&
author) {
    authors.push_back(author);
    saveAuthors();
}

void addPatron(const Patron&
patron) {
    patrons.push_back(patron);
    savePatrons();
}
void addLoan(const Loan& loan) {
    loans.push_back(loan);
    saveLoans();
}
```

// Adding author

6.2 Remove method

```
void removeBook(const string& title)
{                                          // Remove books from a authors based on its title

books.erase(remove_if(books.begin(),
books.end(), [&title](const Book&
book) {
        return book.title == title;
        }), books.end());
    saveBooks();
}
  void removeAuthor(const string&
id) {

                                            // Remove an author from the authors
authors.erase(remove_if(authors.begi        collection based on their author ID
n(), authors.end(), [id](const Author&
author) {
        return author.authorID == id;
        }), authors.end());
    saveAuthors();
}

    void removePatron(int id) {
 patrons.erase(remove_if(patrons.begin(),   // Removes a patron from the patrons
 patrons.end(), [id](const Patron& patron)  collection based on their patron ID (likely an
                {                           integer value)
        return patron.patronID == id;
        }), patrons.end());
    savePatrons();
}

    void removeLoan(int id) {

                                            // Removes a loan record from the loans
loans.erase(remove_if(loans.begin(),        collection based on its loan ID (likely an
loans.end(), [id](const Loan& loan) {       integer value).
        return loan.loanID == id;
        }), loans.end());
    saveLoans();
}
```

6.3 Display Method

```
void displayBooks() const {
    for (const auto& book : books) {
        book.display();
    }
}
```

// This function iterates through the books collection and calls the display function on each Book object, presumably displaying information about the book (title, author, etc.).

```
void displayAuthors() const {
    for (const auto& author :
authors) {
        author.display();
    }
}
```

//This function iterates through the authors collection and calls the display function on each Author object, presumably displaying information about the author (name, ID, etc.).

```
void displayPatrons() const {
    for (const auto& patron :
patrons) {
        patron.display();
    }
}
```

// This function iterates through the patrons collection and calls the display function on each Patron object, presumably displaying information about the patron (name, ID, contact details, etc.).

```
void displayLoans() const {
    for (const auto& loan : loans) {
        loan.display();
    }
}
```

// This function iterates through the loans collection and calls the display function on each Loan object, presumably displaying information about the loan (book borrowed, patron borrowing it, due date, etc.).

6.4 Search Method

```
void searchBookByTitle(const
string& title) const {
    for (const auto& book : books) {
        if (book.title == title) {
            book.display();
            return;
        }
    }
    cout << "Book not found!" <<
endl;
}
```

// This function searches for a book based on its title. If found, it displays the book's information.

```
void searchAuthorByName(const
string& name) const {
    for (const auto& author :
```

// This function searches for an author based on their name. If found, it displays the

```
authors) {
            if (author.name == name) {
                author.display();
                return;
            }
        }
        cout << "Author not found!" <<
endl;
    }

    void searchPatronByID(int id)
const {
        for (const auto& patron :
patrons) {
            if (patron.patronID == id) {
                patron.display();
                return;
            }
        }
        cout << "Patron not found!" <<
endl;
    }
```

author's information.

// This function searches for a patron based on their ID. If found, it displays the patron's information.

6.5 Update Method

```
 void updateBookDetails(const
string& title) {
        for (auto& book : books) {
            if (book.title == title) {
                string newTitle, newAuthor;
                int newYear;
                cout << "Enter new title: ";
                cin >> ws;
                getline(cin, newTitle);
                cout << "Enter new author:
";
                getline(cin, newAuthor);
                cout << "Enter new year: ";
                cin >> newYear;

                book.title = newTitle;
                book.author = newAuthor;
                book.year = newYear;

                saveBooks();
                cout << "Book details
```

// The for loop uses a reference (auto&) to iterate through each element, allowing direct modification within the loop

// assigns these new values to the corresponding member variables of the book object.

// If the user enters the correct value, it will be

```
updated successfully." << endl;
            return;
        }
    }
    cout << "Book not found!" <<
endl;
}

void updateAuthorDetails(const
string& id, const string& newName) {
    for (auto& author : authors) {
        if (author.authorID == id) {
            author.name = newName;
            saveAuthors();
            cout << "Author details
updated successfully." << endl;
            return;
        }
    }
    cout << "Author not found!" <<
endl;
}

void updatePatronDetails(int id,
const string& newName) {
    for (auto& patron : patrons) {
        if (patron.patronID == id) {
            patron.name = newName;
            savePatrons();
            cout << "Patron details
updated successfully." << endl;
            return;
        }
    }
    cout << "Patron not found!" <<
endl;
}
```

updated successfully

7. Getter/Setter

```
vector<Book>& getBooks() {
    return books;
}
vector<Author>& getAuthors() {
return authors;
```

// Returns a reference to a vector containing Book objects.

//Returns a reference to a vector containing Author objects.

```
 vector<Patron>& getPartrons() {
return patrons; }
   vector<Loan>& getLoans() { return
loans; }
};
```

//Returns a reference to a vector containing Patron objects.

//Returns a reference to a vector containing Loan objects.

8. The main function

```
int main() {
   Library library;
   int choice;
```

// Declares an integer variable choice to store the user's menu selection.

8.1 Main Loop (Menu):

```
   while (true) {

       cout <<
"----------------------------";
       cout << "\nLibrary Management
System\n";
       cout << "1. Add Book\n";
       cout << "2. Remove Book\n";
       cout << "3. Add Author\n";
       cout << "4. Remove Author\n";
       cout << "5. Add Patron\n";
       cout << "6. Remove Patron\n";
       cout << "7. Add Loan\n";
       cout << "8. Remove Loan\n";
       cout << "9. Display All Data\n";
       cout << "10. Search Book by
Title\n";
       cout << "11. Search Author by
Name\n";
       cout << "12. Search Patron by
ID\n";
       cout << "13. Update Book
Details\n";
       cout << "14. Update Author
Details\n";
       cout << "15. Update Patron
Details\n";
       cout << "16. Exit\n";
       cout <<
"---------------------------- \n";
       cout << "Enter your choice: ";
       cin >> choice;
```

//A while loop keeps the program running until the user exits.

// The user is prompted to enter their choice.

// A series of if-else if statements handle different user selections

```
 if (choice == 1) {                          // Adding book
        string title, author;
        int year;
        cout << "Enter title: ";
        cin >> ws;
        getline(cin, title);
        cout << "Enter author: ";
        getline(cin, author);
        cout << "Enter year: ";
        cin >> year;
        library.addBook(Book(title,
author, year));
    }
    else if (choice == 2) {
        string title;                           // Remove book
        cout << "Enter Title to
remove: ";
        cin >> ws;
        getline(cin, title);
        library.removeBook(title);
    }
    else if (choice == 3) {
        string name, authorID;
        cout << "Enter authorID: ";             // Adding Author
        cin >> ws;
        getline(cin, authorID);
        cout << "Enter name: ";
        getline(cin, name);

library.addAuthor(Author(authorID,
name));
    }
    else if (choice == 4) {
        string authorID;
        cout << "Enter AuthorID to              // Remove Author
remove: ";
        cin >> ws;
        getline(cin, authorID);

library.removeAuthor(authorID);
    }
    else if (choice == 5) {
        string name;
        int patronID;                           // Add Patron
        cout << "Enter patronID: ";
```

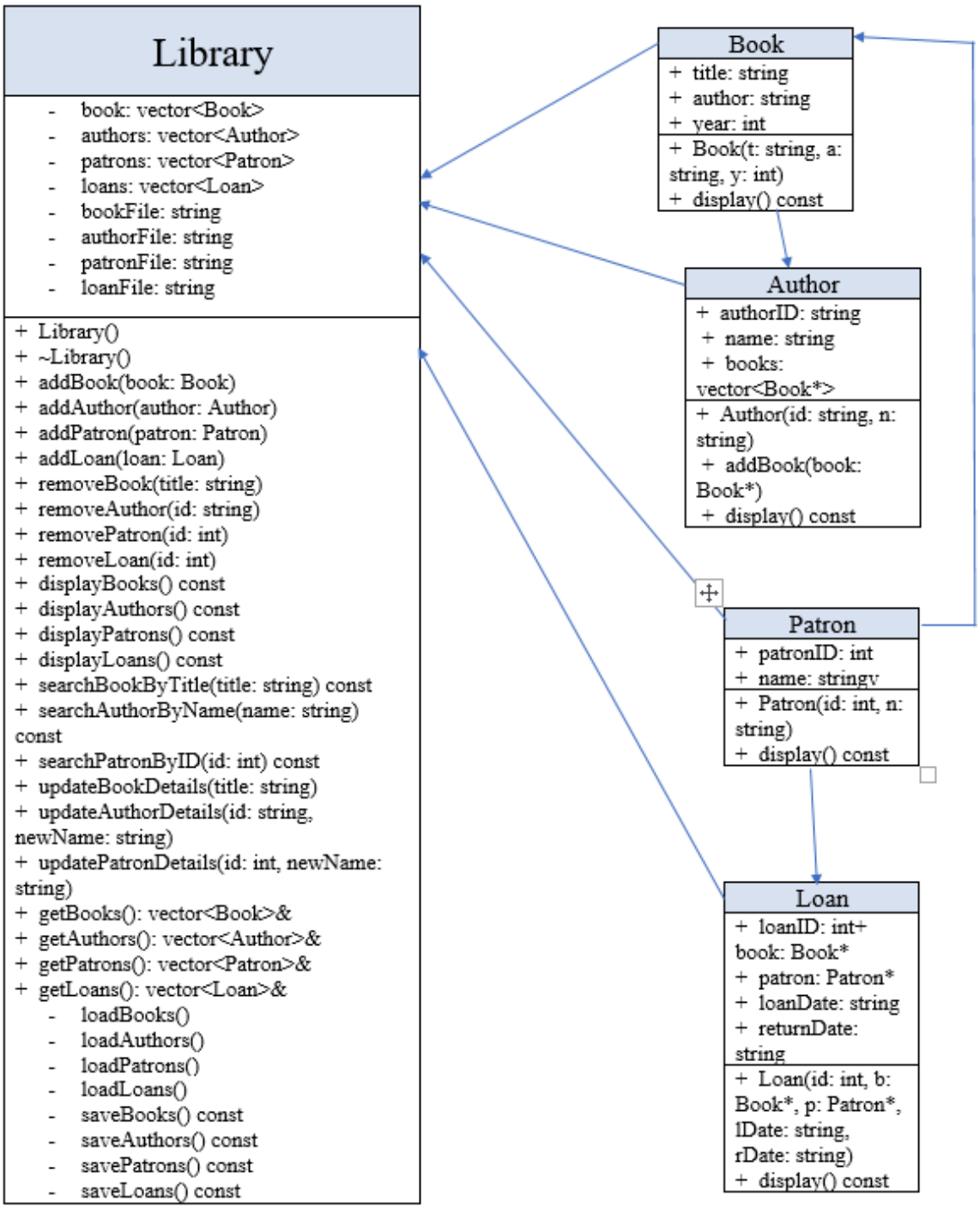
```
            cin >> patronID;
            cin.ignore();
            cout << "Enter name: ";
            getline(cin, name);

library.addPatron(Patron(patronID,
name));
        }
        else if (choice == 6) {
            int patronID;
            cout << "Enter PatronID to
remove: ";                              // Remove Patron
            cin >> patronID;

library.removePatron(patronID);
        }
        else if (choice == 7) {
            int loanID, patronID;
            string title, loanDate,
returnDate;
            cout << "Enter loanID: ";   // Add Loan
            cin >> loanID;
            cin.ignore();
            cout << "Enter book title: ";
            getline(cin, title);
            cout << "Enter patronID: ";
            cin >> patronID;
            cin.ignore();
            cout << "Enter loan date: ";
            getline(cin, loanDate);
            cout << "Enter return date: ";
            getline(cin, returnDate);

            // Find the corresponding book
and patron
            Book* loanBook = nullptr;
            Patron* loanPatron = nullptr;
            for (auto& book :
library.getBooks()) {
                if (book.title == title) {
                    loanBook = &book;
                    break;
                }
            }
            for (auto& patron :
```

```
library.getPartrons()) {
            if (patron.patronID ==
patronID) {
                loanPatron = &patron;
                break;
            }
        }

        if (loanBook && loanPatron)
{

library.addLoan(Loan(loanID,
loanBook, loanPatron, loanDate,
returnDate));
        }
        else {
            cout << "Invalid book title
or patron ID!" << endl;
        }
    }
    else if (choice == 8) {                    // Remove Loan
        int id;
        cout << "Enter Loan ID to
remove: ";
        cin >> id;
        library.removeLoan(id);
    }
    else if (choice == 9) {
        library.displayBooks();
        library.displayAuthors();              // Display all data
        library.displayPatrons();
        library.displayLoans();
    }
    else if (choice == 10) {
        string title;
        cout << "Enter Title: ";
        cin >> ws;
        getline(cin, title);

library.searchBookByTitle(title);
    }                                          // Search book by title
    else if (choice == 11) {
        string author;
        cout << "Enter Author: ";
        cin >> ws;
```

```
            getline(cin, author);

library.searchAuthorByName(author);
        }
        else if (choice == 12) {                    // Search Author by name
            int patronID;
            cout << "Enter ID: ";
            cin >> patronID;

library.searchPatronByID(patronID);
        }
        else if (choice == 13) {                    // Search Patron by ID
            string title;
            cout << "Enter Title to update:
";
            cin >> ws;
            getline(cin, title);

library.updateBookDetails(title);
        }
        else if (choice == 14) {                    // Update Book Details
            string authorID, newName;
            cout << "Enter Author ID to
update: ";
            cin >> ws;
            getline(cin, authorID);
            cout << "Enter New Name: ";
            getline(cin, newName);

library.updateAuthorDetails(authorID,
newName);
        }                                           // Update Author Details
        else if (choice == 15) {
            int patronID;
            string newName;
            cout << "Enter Patron ID to
update: ";
            cin >> patronID;
            cin.ignore();
            cout << "Enter New Name: ";
            getline(cin, newName);

library.updatePatronDetails(patronID,
newName);                                           // Update Patron Details
        }
```

```
        else if (choice == 16) {                // Exit
            break;
        }
        else {
            cout << "Invalid choice,             // An error message is displayed if the user enters
please try again." << endl;                     an invalid choice.
        }
    }

    return 0;                                   // Indicates successful program execution.
}
```

CHAPTER 4: UML CLASS DIAGRAMS



CHAPTER 5: PROGRAM IMPLEMENTATION

1. Main menu

```
PS C:\Users\Admin\Documents\Demo> cd 'c:\Users\Admin\Documents\Demo\output'
PS C:\Users\Admin\Documents\Demo\output> & .\'finalproject.exe'
----------------------------
Library Management System
1. Add Book
2. Remove Book
3. Add Author
4. Remove Author
5. Add Patron
6. Remove Patron
7. Add Loan
8. Remove Loan
9. Display All Data
10. Search Book by Title
11. Search Author by Name
12. Search Patron by ID
13. Update Book Details
14. Update Author Details
15. Update Patron Details
16. Exit
----------------------------
Enter your choice:
```

Main functions in the program's menu

2. Insert information

```
----------------------------
Library Management System
1. Add Book
2. Remove Book
3. Add Author
4. Remove Author
5. Add Patron
6. Remove Patron
7. Add Loan
8. Remove Loan
9. Display All Data
10. Search Book by Title
11. Search Author by Name
12. Search Patron by ID
13. Update Book Details
14. Update Author Details
15. Update Patron Details
16. Exit
----------------------------
Enter your choice: 1
Enter title: Chiec luoc nga
Enter author: Nguyen Quang Sang
Enter year: 1966
----------------------------
```

Add Book

```
----------------------------
Library Management System
1. Add Book
2. Remove Book
3. Add Author
4. Remove Author
5. Add Patron
6. Remove Patron
7. Add Loan
8. Remove Loan
9. Display All Data
10. Search Book by Title
11. Search Author by Name
12. Search Patron by ID
13. Update Book Details
14. Update Author Details
15. Update Patron Details
16. Exit
----------------------------
Enter your choice: 3
Enter authorID: 11
Enter name: Viet Anh
----------------------------
```

Add Author

```
Library Management System
1. Add Book
2. Remove Book
3. Add Author
4. Remove Author
5. Add Patron
6. Remove Patron
7. Add Loan
8. Remove Loan
9. Display All Data
10. Search Book by Title
11. Search Author by Name
12. Search Patron by ID
13. Update Book Details
14. Update Author Details
15. Update Patron Details
16. Exit
----------------------------
Enter your choice: 5
Enter patronID: 01
Enter name: Chi
----------------------------
```

Add Patron

```
Library Management System
1. Add Book
2. Remove Book
3. Add Author
4. Remove Author
5. Add Patron
6. Remove Patron
7. Add Loan
8. Remove Loan
9. Display All Data
10. Search Book by Title
11. Search Author by Name
12. Search Patron by ID
13. Update Book Details
14. Update Author Details
15. Update Patron Details
16. Exit
----------------------------
Enter your choice: 7
Enter loanID: 1234
Enter book title: Lang
Enter patronID: 01
Enter loan date: 29/05
Enter return date: 05/06
Invalid book title or patron ID!
----------------------------
```

Add Loan

3. Remove data

```
----------------------------
Enter your choice: 2
Enter Title to remove: lang
----------------------------
```

Remove Book

```
----------------------------
Enter your choice: 4
Enter AuthorID to remove: 31
----------------------------
```

Remove Author

```
----------------------------
Enter your choice: 6
Enter PatronID to remove: 01
----------------------------
```

Remove Patron

```
----------------------------
Enter your choice: 8
Enter Loan ID to remove: 1234
----------------------------
```

Remove Loan

4. Search Data

```
--------------------------
Enter your choice: 10
Enter Title: Chiec luoc nga
Title: Chiec luoc nga, Author: Nguyen Quang Sang, Year: 1966
--------------------------
```

Search Book by Title

```
----------------------------
Enter your choice: 11
Enter Author: Viet Anh
Author ID: 11, Name: Viet Anh
----------------------------
```

Search Author by Name

```
----------------------------
Enter your choice: 12
Enter ID: 01
Patron ID: 1, Name: Chi
----------------------------
```

Search Patron by ID

5. Update data

```
----------------------------
Enter your choice: 13
Enter Title to update: lang
Enter new title: Làng
Enter new author: Kim Lân
Enter new year: 2002
Book details updated successfully.
----------------------------
```

Update Book Details

```
----------------------------
Enter your choice: 14
Enter Author ID to update: 31
Enter New Name: Oanh
Author details updated successfully.
----------------------------
```

Update Author Details

```
----------------------------
Enter your choice: 15
Enter Patron ID to update: 01
Enter New Name: Nguyễn Quỳnh Chi
Patron details updated successfully.
----------------------------
```

Update Patron Details

6. Exit Program

```
13. Update Book Details
14. Update Author Details
15. Update Patron Details
16. Exit
----------------------------
Enter your choice: 16
PS C:\Users\Admin\Documents\Demo\output>
```

CHAPTER 6: CONCLUSION

Throughout the project, we reach significant milestones such as the successful implementation of code structure and the seamless integration of advanced searching capabilities.

After this project, we could see some advantages and disadvantages of the project.

- Advantage of the project to the library:
    + The project gives the library a better understanding of what they are doing and how many books they have are available to loan and the students who borrow their book.
    + The project gives us a clearer picture of the flow - from the library to the borrower.
    + UML class diagram gives the library a better understanding of the relationships between all the subjects.
    + The database provides a much better way to track students experience
    + It provides a metric to measure it success, like number of book available for borrowing
- Disadvantage of the project to the library:
    + There are few data types that can be entered, for example, there is no option to check when did the student return the book or if the student borrowed/returned books on time or not, and so on.

In this course, we have explored various features of C++ programming and its features. Capabilities that help us develop and manage a successful library management project. We learned how to write code, create data, store data, update records, and build relationships between tables. We also looked at the different types of data that can be stored in a database and how to use these data to create reports. Finally, we discussed the importance of security when using C++ program for our management library project. With these skills, we can confidently take on any project related to project management.

WORK BREAKDOWN STRUCTURE

No	Name	Student ID	Contribution	Comments
1	Đào Thị Phương Anh	22070895	20%	Take part in the creation of the code, draw UML class diagram, participate in researching the fifth chapter of the report.
2	Nguyễn Thái Việt Anh	22070884	20%	Take part in the creation of the code, participate in researching the first and third chapters of the report.
3	Nguyễn Quỳnh Chi	22071081	20%	Take part in the creation of the code, participate in researching the second, third and fourth chapters of the report.
4	Nguyễn Thị Kim Oanh	22070937	20%	Take part in the creation of the code, participate in researching the third (class library) chapter of the report.
5	Nguyễn Đức Hiếu	21070553	20%	Take part in the creation of the code, participate in researching the fourth and fifth chapters of the report.