NAME:KASHIF KHAN

DEPARTMENT:BSCS

SECTION:D

SUBJECT:OOPS

PROJECT: Library Management System

DATE:27/05/202

CODE FOR PROJECT OF LIBRIRY MANIJMENT SYSTEM :

1 • **Book Management**: A class to manage book records.
2 • **Member Management**: A class to manage library members.
3 • **Circulation**: A class to handle borrowing and returning books.
4 • **Overdue Tracking**: A class to track overdue books and calculate fines.
5 • **Reporting**: Methods to generate various reportss a basic implementation:

Here's a basic implementation:

```
 #include <iostream>

#include <vector>
#include <string>
#include <unordered_map>
#include <ctime>
using namespace std;

class Book {
public:
    int id;
    string title;
    string author;
    string genre;
    bool isBorrowed;
    time_t dueDate;

    Book(int id, string title, string author, string genre)
        : id(id), title(title), author(author), genre(genre), isBorrowed(false),
dueDate(0) {}
};

class Member {
public:
    int id;
    string name;
    vector<int> borrowedBooks;

    Member(int id, string name) : id(id), name(name) {}
};

class Library {
private:
    unordered_map<int, Book> books;
    unordered_map<int, Member> members;

public:
    // Book Management
```

```
    void addBook(int id, string title, string author, string genre) {
        books[id] = Book(id, title, author, genre);
    }

    void updateBook(int id, string title, string author, string genre) {
        if (books.find(id) != books.end()) {
            books[id].title = title;
            books[id].author = author;
            books[id].genre = genre;
        }
    }

    void deleteBook(int id) {
        books.erase(id);
    }

    // Search
    vector<Book> searchByTitle(string title) {
        vector<Book> result;
        for (auto &pair : books) {
            if (pair.second.title == title) {
                result.push_back(pair.second);
            }
        }
        return result;
    }

    vector<Book> searchByAuthor(string author) {
        vector<Book> result;
        for (auto &pair : books) {
            if (pair.second.author == author) {
                result.push_back(pair.second);
            }
        }
        return result;
    }

    vector<Book> searchByGenre(string genre) {
        vector<Book> result;
        for (auto &pair : books) {
            if (pair.second.genre == genre) {
                result.push_back(pair.second);
            }
        }
        return result;
    }

    // Book Circulation
    void checkOutBook(int memberId, int bookId, time_t dueDate) {
        if (books.find(bookId) != books.end() && members.find(memberId) !=
members.end()) {
```

```
            books[bookId].isBorrowed = true;
            books[bookId].dueDate = dueDate;
            members[memberId].borrowedBooks.push_back(bookId);
        }
    }

    void returnBook(int memberId, int bookId) {
        if (books.find(bookId) != books.end() && members.find(memberId) !=
members.end()) {
            books[bookId].isBorrowed = false;
            books[bookId].dueDate = 0;
            auto &borrowedBooks = members[memberId].borrowedBooks;
            borrowedBooks.erase(remove(borrowedBooks.begin(),
borrowedBooks.end(), bookId), borrowedBooks.end());
        }
    }

    // Overdue Tracking
    vector<Book> trackOverdueBooks() {
        vector<Book> overdueBooks;
        time_t now = time(0);
        for (auto &pair : books) {
            if (pair.second.isBorrowed && pair.second.dueDate < now) {
                overdueBooks.push_back(pair.second);
            }
        }
        return overdueBooks;
    }

    double calculateFines(int bookId, double fineRatePerDay) {
        if (books.find(bookId) != books.end()) {
            time_t now = time(0);
            double overdueDays = difftime(now, books[bookId].dueDate) / (60 * 60
* 24);
            return overdueDays > 0 ? overdueDays * fineRatePerDay : 0.0;
        }
        return 0.0;
    }

    // Reporting
    vector<Book> getAvailableBooks() {
        vector<Book> availableBooks;
        for (auto &pair : books) {
            if (!pair.second.isBorrowed) {
                availableBooks.push_back(pair.second);
            }
        }
        return availableBooks;
    }

    vector<pair<Member, Book>> getBorrowingHistory() {
```

```
        vector<pair<Member, Book>> history;
        for (auto &memberPair : members) {
            for (int bookId : memberPair.second.borrowedBooks) {
                if (books.find(bookId) != books.end()) {
                    history.push_back({memberPair.second, books[bookId]});
                }
            }
        }
        return history;
    }
};

int main() {
    Library library;

    // Adding books
    library.addBook(1, "The Great Gatsby", "F. Scott Fitzgerald", "Fiction");
    library.addBook(2, "To Kill a Mockingbird", "Harper Lee", "Fiction");

    // Adding members
    library.addMember(1, "John Doe");
    library.addMember(2, "Jane Smith");

    // Checking out and returning books
    time_t dueDate = time(0) + 7 * 24 * 60 * 60; // 1 week from now
    library.checkOutBook(1, 1, dueDate);
    library.returnBook(1, 1);

    // Generating reports
    auto availableBooks = library.getAvailableBooks();
    auto borrowingHistory = library.getBorrowingHistory();

    // Overdue tracking and fine calculation
    auto overdueBooks = library.trackOverdueBooks();
    double fine = library.calculateFines(1, 0.5); // Assuming fine rate is $0.5
per day

    return 0;
}
```

This example provides a starting point and can be further expanded to include more detailed functionality and error handling. Each class and method should be implemented with thorough input validation and error handling for a production system.