**COMSATS University Islamabad, Lahore Campus**

**Block–B, Department of Electrical and Computer Engineering**

**COMSATS Institute of Information Technology, 1.5KM Defence Road, Off Raiwind Road, Lahore**

**Lab Project**

Program Specifications

|  |  |
| --- | --- |
| **Application/ Program name:** | Library Management System |
| **Written by:** | **Name : Muhammad Zawahir Roll number : FA23-BCE-075**  **Name : Muhammad Usman**  **Roll number : FA23-BCE-073** |
|  |  |

|  |
| --- |
| **Purpose or problem definition:** |
| The code aims to simulate a basic library management system with functionalities for adding books, borrowing books, returning books, and calculating fines for late returns. The data related to books, borrowings, and fines is stored in separate text files. |
|  |
| **Program Procedures:** |

|  |
| --- |
| **1. Constants and Structures:**   * The code starts by declaring constants for book limit, seconds in a day, fine per day, and days allowed for borrowing. * Two structures are defined: Book to store book information and Transaction to store transaction details.   **2. Function Prototypes:**   * Prototypes for various functions are declared, indicating their purpose and expected parameters.   **3. Main Function Setup:**   * The main function initializes variables for the library, book count, transaction count, user choice, and transaction array. * It loads existing library data and transaction history from files using loadFromFile and loadTransactionsFromFile functions.   **4. Main Menu Loop:**   * The program enters a do-while loop, displaying the main menu and processing user choices until the user chooses to exit (do { ... } while (choice != 0)).   **5. User Choice and Switch Case:**   * The user is prompted to input a choice, and a switch-case structure handles different menu options. * Choices 1-5 correspond to adding a book, displaying books, borrowing a book, returning a book, and displaying transaction history. * Choice 0 exits the program.   **6. Add Book Function:**   * The addBook function allows the user to input details for a new book, such as name, author, and ID. It checks for ID uniqueness and updates the library accordingly.   **7. Display Books Function:**   * The displayBooks function prints a tabular display of available books in the library, showing their ID, name, author, and availability status.   **8. Borrow Book Function:**   * The borrowBook function displays available books, prompts the user to select a book to borrow, and updates the library and transactions accordingly.   **9. Return Book Function:**   * The returnBook function displays borrowed books, prompts the user to return a book, updates the library, calculates fines if applicable, and removes the corresponding transaction.   **10. Save to File Functions:** - saveToFile and saveTransactionsToFile functions save the current state of the library and transactions to files for persistent storage.  **11. Load from File Functions:** - loadFromFile and loadTransactionsFromFile functions read data from files to initialize the library and transaction history.  **12. Remove Transaction Function:** - The removeTransaction function removes a transaction from the transaction array, adjusting the count.  **13. Calculate Fine Function:** - The calculateFine function determines if a book return is late and calculates the fine accordingly.  **14. Print Transactions from File Function:** - The printTransactionsFromFile function prints the transaction history stored in a file.  **15. Menu Display Function:** - The menu function displays the available menu options to the user.  **16. Book Save to File Function:** - The saveToFile function saves book information to a file named "books.txt".  **17. Transaction Save to File Function:** - The saveTransactionsToFile function saves transaction information to a file named "transactions.txt".  **18. Book Load from File Function:** - The loadFromFile function reads book information from a file named "books.txt".  **19. Transaction Load from File Function:** - The loadTransactionsFromFile function reads transaction information from a file named "transactions.txt".  **20. Remove Transaction Function:** - The removeTransaction function removes a transaction from the transaction array.  **21. Calculate Fine Function:** - The calculateFine function calculates fines based on the return time of a book.  **22. Print Transactions from File Function:** - The printTransactionsFromFile function prints transaction history from a file.  **23. Menu Display Function:** - The menu function displays the menu options for the user.  **24. Add Book Function:** - The addBook function allows the user to add a new book to the library.  **25. Display Books Function:** - The displayBooks function displays the available books in the library.  **26. Borrow Book Function:** - The borrowBook function allows the user to borrow a book from the library.  **27. Return Book Function:** - The returnBook function allows the user to return a borrowed book to the library.  **28. Remove Transaction Function:** - The removeTransaction function removes a transaction from the transaction array.  **29. Calculate Fine Function:** - The calculateFine function calculates fines for late returns.  **30. End of Program:** - The main function returns 0, indicating successful program completion.  **Pseudo Code:**  **1. Constants Declaration:**   * Constants for book limit, seconds in a day, fine per day, and allowed days are declared (**BOOK\_LIMIT**, **SECONDS\_IN\_DAY**, **FINE\_PER\_DAY**, **DAYS\_ALLOWED**).   **2. Structures Declaration:**   * Two structures are defined: **Book** to store book information and **Transaction** to store transaction details (e.g., borrowing and returning books).   **3. Function Prototypes:**   * Prototypes for various functions are declared, indicating their purpose and expected parameters.   **4. Main Function Setup:**   * Variables are initialized for the library (**Book** array), book count, transaction count, user choice, and transaction array (**library**, **bookCount**, **transactionCount**, **choice**, **Ttransaction**).   **5. Load Data from Files:**   * The program loads existing library data and transaction history from files using **loadFromFile** and **loadTransactionsFromFile** functions.   **6. Main Menu Loop:**   * The program enters a do-while loop, displaying the main menu and processing user choices until the user chooses to exit (**do { ... } while (choice != 0)**).   **7. Display Menu Function:**   * The **menu** function prints the main menu options for the user.   **8. User Choice and Switch Case:**   * user is prompted to input a choice, and a switch-case structure handles different menu options.   **9. Add Book Function:**  The The addBook function allows the user to input details for a new book, such as name, author, and ID. It checks for ID uniqueness and updates the library accordingly.  **10. Display Books Function:**  - The displayBooks function prints a tabular display of available books in the library, showing their ID, name, author, and availability status.  **11. Borrow Book Function**:  - The borrowBook function displays available books, prompts the user to select a book to borrow, and updates the library and transactions accordingly.  **12. Return Book Function:**  - The returnBook function displays borrowed books, prompts the user to return a book, updates the library, calculates fines if applicable, and removes the corresponding transaction. |
| **13. Save to File Functions:**   * The **saveToFile** and **saveTransactionsToFile** functions play a crucial role in maintaining persistent storage. These functions save the current state of the library and transactions to files. The **saveToFile** function is responsible for storing book information, while **saveTransactionsToFile** handles the transaction data.   **14. Load from File Functions:**   * The **loadFromFile** and **loadTransactionsFromFile** functions are vital for initializing the library and transaction history. These functions read data from files, specifically "books.txt" and "transactions.txt," to populate the library with existing book information and transaction records.   **15. Remove Transaction Function:**   * The **removeTransaction** function serves to efficiently manage the transaction array. By removing a specific transaction and adjusting the count, this function ensures accurate and streamlined tracking of library activities.   **16. Calculate Fine Function:**   * The **calculateFine** function is responsible for determining whether a book return is late. If so, it calculates the fine based on the return time and the predefined constants for fine per day and days allowed for borrowing.   **17. Print Transactions from File Function:**   * The **printTransactionsFromFile** function offers a convenient way to access and review the transaction history. By reading and displaying data from the "transactions.txt" file, this function provides users with insights into previous interactions with the library.   **18. End of Program:**   * As the user concludes their interaction with the program, the main function gracefully exits by returning 0. This return value indicates the successful completion of the program, ensuring a smooth user experience.   **Flowchart:** |
| **Comments:**   1. **Add a New Book:**    * To add a new book to the library, select option 1 from the main menu. Enter the book details such as name, author, and a unique book ID. 2. **Display Books:**    * Choose option 2 to display all available books in a tabular format. This will provide information on book IDs, names, authors, and availability status. 3. **Borrow a Book:**    * To borrow a book, select option 3. Follow the prompts to view available books, choose one, and enter your name. The system will update the book status and record the transaction. 4. **Return a Book:**    * Returning a book is easy with option 4. Simply enter the book ID you want to return, and the system will update availability, calculate fines (if any), and remove the corresponding transaction. 5. **Transaction History:**    * Option 5 allows you to view the transaction history, displaying details such as book ID, customer name, transaction type, and borrowing date. 6. **Exit:**    * To exit the system, choose option 0. Your data is automatically saved for future use. |

**Library Management System**

#include <iostream>

#include <iomanip>

#include <fstream>

#include <ctime>

using namespace std;

// Declaring constants

const int BOOK\_LIMIT = 100;

const int SECONDS\_IN\_DAY = 86400;

const int FINE\_PER\_DAY = 2;

const int DAYS\_ALLOWED = 1;

// defining structures

struct Book

{

    int BookID;

    string bookName;

    string bookAuthor;

    bool available;

};

struct Transaction

{

    int BookID;

    string username;

    string transactionType;

    time\_t borrowingTime;

    time\_t returningTime;

};

// Function Prototypes

void menu();

void addBook(Book library[], int &bookCount);

void displayBooks(Book library[], int &bookCount);

void borrowBook(Book library[], int &bookCount, Transaction Ttransaction[], int &transactionCount);

void returnBook(Book library[], int &bookCount, Transaction Ttransaction[], int &transactionCount);

void saveToFile(Book library[], int &bookCount);

void saveTransactionsToFile(Transaction Ttransaction[], int transactionCount);

void loadFromFile(Book library[], int &bookCount);

void loadTransactionsFromFile(Transaction Ttransaction[], int &transactionCount);

void removeTransaction(Transaction Ttransaction[], int &transactionCount, int index);

void calculateFine(Transaction &transaction);

void printTransactionsFromFile(Transaction Ttransaction[], int transactionCount);

int main()

{

    Book library[BOOK\_LIMIT];

    int bookCount = 0;        // starting index of the library array

    int transactionCount = 0; // starting index of transaction array

    int choice;

    Transaction Ttransaction[BOOK\_LIMIT];

    loadFromFile(library, bookCount);

    loadTransactionsFromFile(Ttransaction, transactionCount); // calling functions so that the data saved in files can be extracted

    do

    {

        menu();

        cout << "Enter you choice: "; // asking the user to select what action they want to perform

        cin >> choice;

        switch (choice) // switch case statements that will call the function that the user wants

        {

        case 1:

            addBook(library, bookCount);

            break;

        case 2:

            displayBooks(library, bookCount);

            cout << "\nPress enter to continue";

            cin.ignore();

            cin.get();

            break;

        case 3:

            borrowBook(library, bookCount, Ttransaction, transactionCount);

            cout << "\nPress enter to continue";

            cin.get();

            break;

        case 4:

            returnBook(library, bookCount, Ttransaction, transactionCount);

            cout << "\nPress enter to continue";

            cin.ignore();

            cin.get();

            break;

        case 5:

            printTransactionsFromFile(Ttransaction, transactionCount);

            cout << "\nPress enter to continue";

            cin.ignore();

            cin.get();

            break;

        case 0:

            cout << "\nGoodBye!" << endl;

            return 0;

        default:

            cout << "\nInvalid input, please enter again.\nPress enter to continue"; // If the user enter invalid input like anything other than from 0 to 5 is invalid or if the user entered the wrong dataType variable

            cin.ignore();

            cin.get();

        }

        saveToFile(library, bookCount);

        saveTransactionsToFile(Ttransaction, transactionCount);

    } while (choice != 0);

    return 0;

}

void menu() // displaying all the available options

{

    cout << "\n\tlibrary Management System\t\n"

         << endl;

    cout << "What would you like to do?\n";

    cout << "1. Add a new book" << endl;

    cout << "2. Display all book in the library" << endl;

    cout << "3. Borrow a Book" << endl;

    cout << "4. Return a Book" << endl;

    cout << "5. Display Transaction History" << endl;

    cout << "0. Exit" << endl

         << endl;

    return;

}

void addBook(Book library[], int &bookCount)

{

    if (bookCount < BOOK\_LIMIT)

    {

        bool ID = false; // flag is initially set to false

        Book newBook;

        // taking input about book and details

        cout << "Enter Book Name:";

        cin.ignore();

        getline(cin, newBook.bookName);

        cout << "Enter Author's Name:";

        getline(cin, newBook.bookAuthor);

        do

        {

            cout << "Enter book ID:";

            cin >> newBook.BookID;

            ID = true;

            for (int i = 0; i < bookCount; i++)

            {

                if (newBook.BookID == library[i].BookID)

                {

                    cout << "This book ID already exists" << endl; // if another book with the same ID already exists, it should not store another book in the same ID

                    ID = false;

                    break;

                }

            }

        } while (!ID);

        newBook.available = true;       // since the book has just been added, so it is available

        library[bookCount++] = newBook; // for the next iteration

        cout << "\nBook successfully added to library." << endl;

    }

    else

    {

        cout << "\nLimit reached. The book can not be added to the library." << endl;

    }

    cout << "\nPress enter to continue";

    cin.ignore();

    cin.get();

}

void displayBooks(Book library[], int &bookCount) // display all the available books

{

    if (bookCount > 0) // if any books are in the library, then the loop will run

    {

        cout << "----------------------------------------------------------------------------------------------" << endl;

        cout << setw(10) << "ID" << setw(25) << "Book Name" << setw(25) << "Author Name" << setw(30) << "Availability Status" << endl; // for a presentable display

        string status;

        for (int i = 0; i < bookCount; i++) // loop will continue until the last book in the library

        {

            if (library[i].available == true) // If the bool variable os status is true, then it is available as we have done earlier and vice versa

            {

                status = "Available";

            }

            else

            {

                status = "Borrowed";

            }

            cout << "----------------------------------------------------------------------------------------------" << endl;

            cout << setw(10) << library[i].BookID << setw(25) << library[i].bookName << setw(25) << library[i].bookAuthor << setw(25) << status << endl;

        }

        cout << "----------------------------------------------------------------------------------------------" << endl;

    }

    else

    {

        cout << "\nNo books in the library to display." << endl; // if no books, nothing to display

    }

    return;

}

// borrow a book from the library

void borrowBook(Book library[], int &bookCount, Transaction Ttransaction[], int &transactionCount)

{

    int bookID = 0;

    if (bookCount > 0)

    {

        cout << "----------------------------------------------------------------------------------------------" << endl;

        cout << setw(10) << "ID" << setw(25) << "Book Name" << setw(25) << "Author Name" << setw(30) << "Availability Status" << endl;

        string status;

        // showing the user all the books so that they know what they want to borrow

        for (int i = 0; i < bookCount; i++)

        {

            if (library[i].available == true)

            {

                status = "Available";

            }

            else

            {

                status = "Borrowed";

            }

            cout << "----------------------------------------------------------------------------------------------" << endl;

            cout << setw(10) << library[i].BookID << setw(25) << library[i].bookName << setw(25) << library[i].bookAuthor << setw(25) << status << endl;

        }

        cout << "----------------------------------------------------------------------------------------------" << endl;

        cout << "Enter ID of the book you want to borrow: ";

        cin >> bookID;

        for (int i = 0; i < bookCount; i++)

        {

            if (library[i].BookID == bookID && library[i].available)

            {

                char confirm;

                cout << "\nDo you want to borrow the book \"" << library[i].bookName << "\"? (y/n) "; // confirming from the user if they want the same book by telling them the name of the book

                cin >> confirm;

                if (confirm == 'y' || confirm == 'Y')

                {

                    Transaction transaction;

                    transaction.BookID = bookID;

                    cout << "Enter your name: ";

                    cin.ignore();

                    getline(cin, transaction.username);

                    transaction.transactionType = "Borrow";

                    transaction.borrowingTime = time(0);

                    library[i].available = false;

                    Ttransaction[transactionCount++] = transaction;

                    cout << "\nBook borrowed successfully. Please return it in time!" << endl;

                }

            }

            else if (library[i].BookID < bookID || library[i].BookID > bookID)

            {

                // Empty so that the for loop works properly

            }

            else

            {

                cout << "\nBook not available or already borrowed." << endl;

            }

        }

    }

    else

    {

        cout << setw(20) << "\nNo book available" << endl;

    }

}

// returning a borrowed book

void returnBook(Book library[], int &bookCount, Transaction Ttransaction[], int &transactionCount)

{

    int bookID;

    if (transactionCount > 0)

    {

        cout << "----------------------------------------------------------------------------------------------" << endl; // for readability

        cout << setw(10) << "ID" << setw(25) << "Book Name" << setw(25) << "Author Name" << setw(30) << "Availability Status" << endl;

        string status;

        for (int i = 0; i < bookCount; i++)

        {

            // if the book is not borrowed it will show "Available" in status column and if it is borrowed, it shows "Borrowed".

            if (library[i].available == true)

            {

                status = "Available";

            }

            else

            {

                status = "Borrowed";

            }

            cout << "----------------------------------------------------------------------------------------------" << endl;

            cout << setw(10) << library[i].BookID << setw(25) << library[i].bookName << setw(25) << library[i].bookAuthor << setw(25) << status << endl;

        }

        cout << "----------------------------------------------------------------------------------------------" << endl;

        cout << "\nEnter ID of the book you want to return: ";

        cin >> bookID;

        // loop for calculating transaction

        for (int i = 0; i < transactionCount; i++)

        {

            if (Ttransaction[i].BookID == bookID)

            {

                Transaction transaction = Ttransaction[i];

                for (int j = 0; j < bookCount; j++)

                {

                    if (library[j].BookID == bookID)

                    {

                        library[j].available = true;

                        break;

                    }

                }

                transaction.returningTime = time(0);

                calculateFine(transaction);

                removeTransaction(Ttransaction, transactionCount, i);

                cout << "\nBook returned successfully" << endl;

                return;

            }

        }

        cout << "\nBook not found in transactions" << endl;

    }

    else

    {

        cout << setw(20) << "No book available for returning" << endl;

    }

}

// Print transactions from file

void printTransactionsFromFile(Transaction Ttransaction[], int transactionCount)

{

    ifstream inFile("transactions.txt");

    if (!inFile)

    {

        cout << "Starting with an empty transaction history." << endl;

        return;

    }

    if (transactionCount > 0)

    {

        cout << setw(65) << "TRANSACTION HISTORY" << endl;

        cout << setw(20) << "Book ID " << setw(15) << "Customer Name" << setw(20) << "Transaction Type" << setw(23) << "Borrowing Date" << endl;

        cout << setw(20) << "---------------------------------------------------------------------------------------" << endl;

        for (int i = 0; i < transactionCount; i++)

        {

            inFile >> Ttransaction[i].BookID >> Ttransaction[i].username >> Ttransaction[i].transactionType >> Ttransaction[i].borrowingTime >> Ttransaction[i].returningTime;

            tm tm\_borrow = \*localtime(&Ttransaction[i].borrowingTime);

            char borrowingTimeString[20];

            strftime(borrowingTimeString, sizeof(borrowingTimeString), "%Y-%m-%d %H:%M:%S", &tm\_borrow);

            cout << setw(17) << Ttransaction[i].BookID << setw(15) << Ttransaction[i].username << setw(18) << Ttransaction[i].transactionType << setw(31) << borrowingTimeString << endl;

        }

        inFile.close();

    }

    else

    {

        cout << "\nNo Transactions were found." << endl;

    }

}

// Save books to file

void saveToFile(Book library[], int &bookCount) // Saving the data in a text file, so we can use the data even when we have exited the program and we are running it again

{

    ofstream outFile;

    outFile.open("books.txt"); // declaring a text file to write, this will make a text file if not already made and if it is already made, it will write in that file

    if (outFile.is\_open()) // if there is no error in opening the file

    {

        outFile << bookCount;

        for (int i = 0; i < bookCount; ++i)

        {

            // storing data in the text file

            outFile << library[i].bookName << endl

                    << library[i].bookAuthor << endl

                    << library[i].BookID << endl

                    << library[i].available << endl;

        }

        outFile.close();

    }

    else // if there is an error opening file

    {

        cout << "Error: Cannot open file" << endl;

    }

    return;

}

// Save transactions to file

void saveTransactionsToFile(Transaction Ttransaction[], int transactionCount)

{

    ofstream outFile("transactions.txt");

    if (!outFile)

    {

        cout << "Error opening file for writing." << endl;

        return;

    }

    for (int i = 0; i < transactionCount; i++)

    {

        outFile << Ttransaction[i].BookID << " "

                << Ttransaction[i].username << " "

                << Ttransaction[i].transactionType << " "

                << (Ttransaction[i].borrowingTime) << " "

                << Ttransaction[i].returningTime << endl;

    }

    outFile.close();

}

// Load books from file

void loadFromFile(Book library[], int &bookCount) // To read the data that we saved in the file in the previous function

{

    ifstream inFile;

    inFile.open("books.txt"); // declaring a text file to read, this wil read from the text file that we created in the above function

    if (inFile.is\_open())     // if ther is no error in opening the file

    {

        inFile >> ws >> bookCount >> ws;

        for (int i = 0; i < bookCount; i++)

        {

            // reading data from file

            getline(inFile, library[i].bookName, '\n');

            getline(inFile, library[i].bookAuthor, '\n');

            inFile >> library[i].BookID >> ws >> library[i].available;

            inFile.ignore();

        }

        inFile.close();

    }

    else // if error while opening file

    {

        cout << "Error: Could not load data from file." << endl;

    }

    return;

}

// Load transactions from file

void loadTransactionsFromFile(Transaction Ttransaction[], int &transactionCount)

{

    ifstream inFile("transactions.txt");

    if (!inFile)

    {

        cout << "Starting with an empty transaction history." << endl;

        return;

    }

    for (; transactionCount < BOOK\_LIMIT && inFile >> Ttransaction[transactionCount].BookID >> Ttransaction[transactionCount].username >> Ttransaction[transactionCount].transactionType >> Ttransaction[transactionCount].borrowingTime >> Ttransaction[transactionCount].returningTime; ++transactionCount)

    {

    }

    inFile.close();

}

// removing transactions

void removeTransaction(Transaction Ttransaction[], int &transactionCount, int index)

{

    for (int i = index; i < transactionCount - 1; i++)

    {

        Ttransaction[i] = Ttransaction[i + 1];

    }

    transactionCount--;

}

// fine for late returning

void calculateFine(Transaction &transaction)

{

    int totalDays = (transaction.returningTime - transaction.borrowingTime) / SECONDS\_IN\_DAY;

    if (totalDays > 1)

    {

        int fine = (totalDays - DAYS\_ALLOWED) \* FINE\_PER\_DAY;

        cout << "\nPay Fine for late return: $" << fine << endl;

    }

    else

    {

        cout << "\nNo fine!You have returned the book in time." << endl;

    }

}