```
#include <iostream>
#include <string>
#include <fstream>
#include <algorithm> //use in funtion all_in

using namespace std;

struct Book
{
   int id;
   string title, author, genre, status;
   Book *next;
};

struct Member
{
   // int id;
   string memberID, memberName, memberContact, memberStatus;
   Member *next;
};

struct Borrow
{
   int bookId, memberId;
   string title, author, genre, statu, name, contact, status;
   Borrow *next;
};

class BookManagement
{
public:
   Book *head = NULL;
   void insertBook();
   void updateBook();
   void deleteBook();
   void searchBook();
   void showBook();
   void menu();
```

```
};

class MemberManagement
{
public:
   Member *head = NULL;
   void insertMember();
   void updateMember();
   void deleteMember();
   void searchMember();
};

void BookManagement::menu()
{
   MemberManagement memberManagement;
   int choice;
   while (true)
   {
      cout << "========================================\n";
      cout << "==    LIBRARY MANAGEMENT SYSTEM    ==\n";
      cout << "========================================\n";
      cout << "==                                  ==\n";
      cout << "==  1. Register New Book        ==\n";
      cout << "==  2. Update Book Information    ==\n";
      cout << "==  3. Delete Book Record       ==\n";
      cout << "==  4. Search Book              ==\n";
      cout << "==  5. Show All Book            ==\n";
      cout << "==  6. Register New Member      ==\n";
      cout << "==  7. Update Member information  ==\n";
      cout << "==  8. Delete Member information  ==\n";
      cout << "==  9. Search Member            ==\n";
      cout << "==  0. Exit                     ==\n";
      cout << "========================================\n";

      cout << "Enter your choice: ";
      cin >> choice;

      switch (choice)
```

```
    {
    case 1:
      insertBook();
      break;
    case 2:
      updateBook();
      break;
    case 3:
      deleteBook();
      break;
    case 4:
      searchBook();
      break;
    case 5:
     showBook();
     break;
    case 6:
      memberManagement.insertMember();
      break;
    case 7:
      memberManagement.updateMember();
      break;
    case 8:
      memberManagement.deleteMember();
      break;
    case 9:
      memberManagement.searchMember();
      break;
    case 0:
      return; // Exit the function
    default:
      cout << "Invalid Choice!" << endl;
    }
  }
}

void BookManagement::insertBook()
{
```

```
ofstream outfile;                // declare output file
outfile.open("books.txt", ios::app); // open output file

Book *newBook = new Book;

cout << "\nEnter Book ID: "; // enter book id
cin >> newBook->id;

cout << "Enter Book Title: "; // enter book title
cin.ignore();
getline(cin, newBook->title);

cout << "Enter Book Author: "; // enter book author
getline(cin, newBook->author);

cout << "Enter Book Genre: "; // enter book genre
getline(cin, newBook->genre);

newBook->status = "available"; // set book status as available

// input into books.txt file
outfile << "Book ID: " << newBook->id << endl;
outfile << "Book Title: " << newBook->title << endl;
outfile << "Book Author: " << newBook->author << endl;
outfile << "Book Genre: " << newBook->genre << endl;
outfile << "Book Status: " << newBook->status << endl;

newBook->next = NULL;

if (head == NULL)
{
   head = newBook;
}
else
{
   Book *ptr = head;

   while (ptr->next != NULL)
```

```
    {
      ptr = ptr->next;
    }
    ptr->next = newBook;
  }
  cout << "\nBook registered successfully!" << endl;
}

void BookManagement::updateBook()
{
  int id;
  cout << "\nEnter Book ID to Update: ";
  cin >> id;

  ifstream infile("books.txt");  // Open file for reading
  ofstream tempFile("temp.txt"); // Temporary file to store updated records

  if (!infile || !tempFile)
  {
    cout << "Error opening file!" << endl;
    return;
  }

  bool bookFound = false;
  string line, updatedTitle, updatedAuthor, updatedGenre, updatedStatus;
  int currentId;
  char choose;

  while (getline(infile, line)) // Read each line
  {
    if (line.rfind("Book ID: ", 0) == 0) // Check if line starts with "Book ID: "
    {
      currentId = stoi(line.substr(9)); // Extract ID from the line

      if (currentId == id)
      {
        // Book found
        bookFound = true;
```

```
cout << "\nBook ID: " << currentId << endl;

// Get updated details from user
cout << "Enter Updated Book Title: ";
cin.ignore();
getline(cin, updatedTitle);

cout << "Enter Updated Book Author: ";
getline(cin, updatedAuthor);

cout << "Enter Updated Book Genre: ";
getline(cin, updatedGenre);

cout << "Enter Updated Book Status (available(A)/not available(N)): ";
cin >> choose;

if (choose == 'a' || choose == 'A')
{
   updatedStatus = "available";
}
else if (choose == 'n' || choose == 'N')
{
   updatedStatus = "not available";
}
else
{
   cout << "Invalid!" << endl;
   return;
}

// Write updated book details to temp file
tempFile << "Book ID: " << currentId << endl;
tempFile << "Book Title: " << updatedTitle << endl;
tempFile << "Book Author: " << updatedAuthor << endl;
tempFile << "Book Genre: " << updatedGenre << endl;
tempFile << "Book Status: " << updatedStatus << endl;
```

```
            // Skip the next 4 lines in the file (as they belong to the current book)
            for (int i = 0; i < 4; i++)
               getline(infile, line);

            continue; // Skip to the next iteration
         }
      }
      // Write the current line to the temp file if it doesn't belong to the book being updated
      tempFile << line << endl;
   }

   infile.close();
   tempFile.close();

   if (bookFound)
   {
      // Replace original file with updated temp file
      remove("books.txt");
      rename("temp.txt", "books.txt");
      cout << "\nBook Record Updated Successfully!" << endl;
   }
   else
   {
      remove("temp.txt"); // Remove temp file if no update was made
      cout << "\nBook Not Found!" << endl;
   }
}

void BookManagement::deleteBook()
{
   int id;
   cout << "\nEnter Book ID to Delete: ";
   cin >> id;

   ifstream infile("books.txt");  // Open the original file for reading
   ofstream tempFile("temp.txt"); // Create a temporary file for updated content

   if (!infile || !tempFile)
```

```
{
   cout << "Error opening file!" << endl;
   return;
}

bool bookFound = false;
string line;
int currentId;

while (getline(infile, line)) // Read each line
{
   if (line.rfind("Book ID: ", 0) == 0) // Check if the line starts with "Book ID: "
   {
      currentId = stoi(line.substr(9)); // Extract the book ID

      if (currentId == id)
      {
         // Skip this book's lines (5 lines for each book)
         bookFound = true;
         for (int i = 0; i < 4; i++) // Skip the next 4 lines
            getline(infile, line);
         continue; // Skip to the next iteration
      }
   }

   // Write all lines except the ones to be deleted
   tempFile << line << endl;
}

infile.close();
tempFile.close();

if (bookFound)
{
   // Replace original file with the updated temporary file
   remove("books.txt");
   rename("temp.txt", "books.txt");
   cout << "\nBook Record Deleted Successfully!" << endl;
```

```
    }
    else
    {
        remove("temp.txt"); // Remove temp file if no book was found
        cout << "\nBook Not Found!" << endl;
    }
}

void BookManagement::searchBook()
{
    int id;
    cout << "\nEnter Book ID to Search: ";
    cin >> id;

    ifstream infile("books.txt"); // Open the file for reading

    if (!infile)
    {
        cout << "Error opening file!" << endl;
        return;
    }

    bool bookFound = false;
    string line;
    int currentId;
    string title, author, genre, status;

    while (getline(infile, line)) // Read each line
    {
        if (line.rfind("Book ID: ", 0) == 0) // Check if the line starts with "Book ID: "
        {
            currentId = stoi(line.substr(9)); // Extract the book ID

            if (currentId == id)
            {
                // Book found, read its details
                bookFound = true;
                cout << "\n============================\n";
```

```
            cout << "   List of book\n";
            cout << "============================\n";

            cout << "\nBook ID: " << currentId << endl;
            getline(infile, line);
            title = line.substr(12); // Extract title
            cout << "Book Title: " << title << endl;

            getline(infile, line);
            author = line.substr(13); // Extract author
            cout << "Book Author: " << author << endl;

            getline(infile, line);
            genre = line.substr(12); // Extract genre
            cout << "Book Genre: " << genre << endl;

            getline(infile, line);
            status = line.substr(13); // Extract status
            cout << "Book Status: " << status << endl;

            break;
        }
      }
   }

   infile.close();

   if (!bookFound)
   {
      cout << "\nBook Not Found!" << endl;
   }
}

void BookManagement::showBook()
{
   ifstream infile("books.txt"); // Open the file for reading

   if (!infile)
```

```
    {
        cout << "Error opening file or no records found!" << endl;
        return;
    }

    string line;
    bool hasBooks = false;

    cout << "\n============================\n";
    cout << "   List of All Books\n";
    cout << "============================\n";

    while (getline(infile, line))
    {
        hasBooks = true;
        cout << line << endl;

        // Check for an empty line between books for better readability
        if (line.empty())
        {
            cout << "----------------------------\n";
        }
    }

    infile.close();

    if (!hasBooks)
{
cout << "\nNo book records available.\n";
}
}

//
_____________________________________________________________________member_____________________________________
__//

void MemberManagement::insertMember()
{
```

```
ofstream outfile;                // declare output file
outfile.open("member.txt", ios::app); // open output file

Member *newMember = new Member;

cout << "Enter Member ID: "; // enter member ID
cin.ignore();
getline(cin, newMember->memberID);

cout << "\nEnter Member Name: "; // enter member name
cin >> newMember->memberName;

cout << "Enter Contact Number: "; // enter contact number
cin.ignore();
getline(cin, newMember->memberContact);

cout << "Enter Updated Member Status (Aktif(A)/Unaktif(U)): ";
char choose;
cin >> choose;
if (choose == 'a' || choose == 'A')
   newMember->memberStatus = "Aktif";
else if (choose == 'u' || choose == 'U')
   newMember->memberStatus = "Unaktif";
else
{
   cerr << "Invalid input for status!" << endl;
   return;
}

// newMember->memberStatus = "Active Member"; // set status as available

// input into member.txt file
outfile << "Member ID: " << newMember->memberID << endl;
outfile << "Member Name: " << newMember->memberName << endl;
outfile << "Member Contact: " << newMember->memberContact << endl;
outfile << "Member Status: " << newMember->memberStatus << endl;

newMember->next = NULL;
```

```
    if (head == NULL)
    {
        head = newMember;
    }
    else
    {
        Member *ptr = head;

        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        ptr->next = newMember;
    }
    cout << "\nMember registered successfully!" << endl;
}

void MemberManagement::updateMember()
{
    string memberID;
    cout << "\nEnter Member ID to Update: ";
    cin.ignore();
    getline(cin, memberID);

    ifstream infile("member.txt");
    ofstream tempFile("temp.txt");

    if (!infile.is_open() || !tempFile.is_open())
    {
        cerr << "Error: Unable to open file!" << endl;
        return;
    }

    bool memberFound = false;
    string line, updatedMemberName, updatedMemberContact, updatedMemberStatus;
    char choose;
```

```
cout << "\nStarting to process the file...\n";

while (getline(infile, line))
{
   // Check if the current line is a "Member ID" line
   if (line.rfind("Member ID: ", 0) == 0)
   {
      string currentId = line.substr(11); // Extract ID after "Member ID: "

      if (currentId == memberID)
      {
         memberFound = true;

         // Get updated details from the user
         cout << "\nMember ID: " << currentId << endl;
         cout << "Enter Updated Member Name: ";
         getline(cin, updatedMemberName);

         cout << "Enter Updated Member Contact: ";
         getline(cin, updatedMemberContact);

         cout << "Enter Updated Member Status (Aktif(A)/Unaktif(U)): ";
         cin >> choose;

         if (choose == 'a' || choose == 'A')
            updatedMemberStatus = "Aktif";
         else if (choose == 'u' || choose == 'U')
            updatedMemberStatus = "Unaktif";
         else
         {
            cerr << "Invalid input for status!" << endl;
            infile.close();
            tempFile.close();
            remove("temp.txt");
            return;
         }

         // Write updated member details to temp file
```

```
            tempFile << "Member ID: " << currentId << endl;
            tempFile << "Member Name: " << updatedMemberName << endl;
            tempFile << "Member Contact: " << updatedMemberContact << endl;
            tempFile << "Member Status: " << updatedMemberStatus << endl;

            // Skip the rest of this member's details in the input file
            while (getline(infile, line) && !line.empty() && line.rfind("Member ID: ", 0) != 0)
            {
                // Do nothing, just skip
            }

            // If the next line starts with "Member ID:", process it next iteration
            if (!line.empty() && line.rfind("Member ID: ", 0) == 0)
            {
                tempFile << line << endl;
            }
            continue;
        }
    }

    // Write non-matching lines to the temp file
    tempFile << line << endl;
}

infile.close();
tempFile.close();

if (memberFound)
{
    // Replace original file with the updated temp file
    remove("member.txt");
    rename("temp.txt", "member.txt");
    cout << "\nMember Record Updated Successfully!" << endl;
}
else
{
    remove("temp.txt"); // Remove temp file if no update was made
    cout << "\nMember Not Found!" << endl;
```

```
    }
}

void MemberManagement::deleteMember()
{
    string memberID;
    cout << "\nEnter Member ID to Delete: ";
    cin.ignore();
    getline(cin, memberID);

    ifstream infile("member.txt");  // Open the original file for reading
    ofstream tempFile("temp.txt"); // Create a temporary file for updated content

    if (!infile || !tempFile)
    {
        cout << "Error opening file!" << endl;
        return;
    }

    bool memberFound = false;
    string line;

    while (getline(infile, line)) // Read each line
    {
        if (line.rfind("Member ID: ", 0) == 0) // Check if the line starts with "Member ID: "
        {
            string currentId = line.substr(11); // Extract the member ID

            if (currentId == memberID)
            {
                // Skip this member's lines (4 lines for each member)
                memberFound = true;
                for (int i = 0; i < 3; i++) // Skip the next 3 lines
                    getline(infile, line);
                continue; // Skip to the next iteration
            }
        }
```

```
        // Write all lines except the ones to be deleted
        tempFile << line << endl;
    }

    infile.close();
    tempFile.close();

    if (memberFound)
    {
        // Replace original file with the updated temporary file
        remove("member.txt");
        rename("temp.txt", "member.txt");
        cout << "\nMember Record Deleted Successfully!" << endl;
    }
    else
    {
        remove("temp.txt"); // Remove temp file if no member was found
        cout << "\nMember Not Found!" << endl;
    }
}

void MemberManagement::searchMember()
{
    string memberID;
    cout << "\nEnter Member ID to Search: ";
    cin.ignore();
    getline(cin, memberID);

    ifstream infile("member.txt"); // Open the file for reading

    if (!infile)
    {
        cout << "Error opening file!" << endl;
        return;
    }

    bool memberFound = false;
    string line;
```

```
while (getline(infile, line)) // Read each line
{
   if (line.rfind("Member ID: ", 0) == 0) // Check if the line starts with "Member ID: "
   {
      string currentId = line.substr(11); // Extract the member ID (after "Member ID: ")

      if (currentId == memberID) // Compare as strings
      {
         // Member found, read its details
         cout << "\n============================\n";
         cout << "   List of member\n";
         cout << "============================\n";
         memberFound = true;

         cout << "\nMember ID: " << currentId << endl;

         getline(infile, line);
         string memberName = line.substr(12); // Extract Name
         cout << "Member Name: " << memberName << endl;

         getline(infile, line);
         string memberContact = line.substr(16); // Extract Contact
         cout << "Member Contact: " << memberContact << endl;

         getline(infile, line);
         string memberStatus = line.substr(15); // Extract Status
         cout << "Member Status: " << memberStatus << endl;

         break;
      }
   }
}

infile.close();

if (!memberFound)
{
```

```
      cout << "\nMember Not Found!" << endl;
   }
}

int main()
{
BookManagement bookManagement;
MemberManagement memberManagement;
bookManagement.menu();
return 0;
}
```