课 程 设 计 报 告

课程名称：计算机程序设计基础（2）

|  |  |
| --- | --- |
| 班级 | 无46 |
| 学号 | 2024010599 |
| 姓名 | 雷涛 |

2025年7月6日

1.**系统需求分析**（图书管理系统）

**目的**：实现图书管理的数字化，提高借阅效率、减少人工错误。

**意义**：简化图书查询、借阅、归还流程。提供数据统计功能辅决策（如热门书籍分析）。为图书馆节省管理成本。

**核心功能**：

**用户信息管理**

功能：增删改查用户信息（姓名、ID、借阅历史）。

输入：用户ID、姓名等；输出：用户详情列表。

**图书信息管理**

功能：增删改查图书信息（书名、作者、ISBN、借阅状态），支持**模糊查询**（如输入“编”匹配“编程”“编译原理”）。

输入：书名/作者关键字；输出：匹配的图书列表。

**借还书记录管理**

功能：记录借书/还书时间，查询用户或图书的借阅历史。

输入：用户ID或图书ID；输出：借阅记录表。

**统计分析功能**

功能：分析最受欢迎图书、借阅最多用户、借阅趋势（按月/年）。

输出：统计图表或报告（控制台可输出表格/简易柱状图）。

**特殊情况处理**

功能：校验非法输入（如无效ID、空字段）、处理边界情况（如重复借书）。

**扩展功能建议**

登录/权限管理（管理员 vs 普通用户）。

借阅规则限制（如最多借5本）

2.总体设计

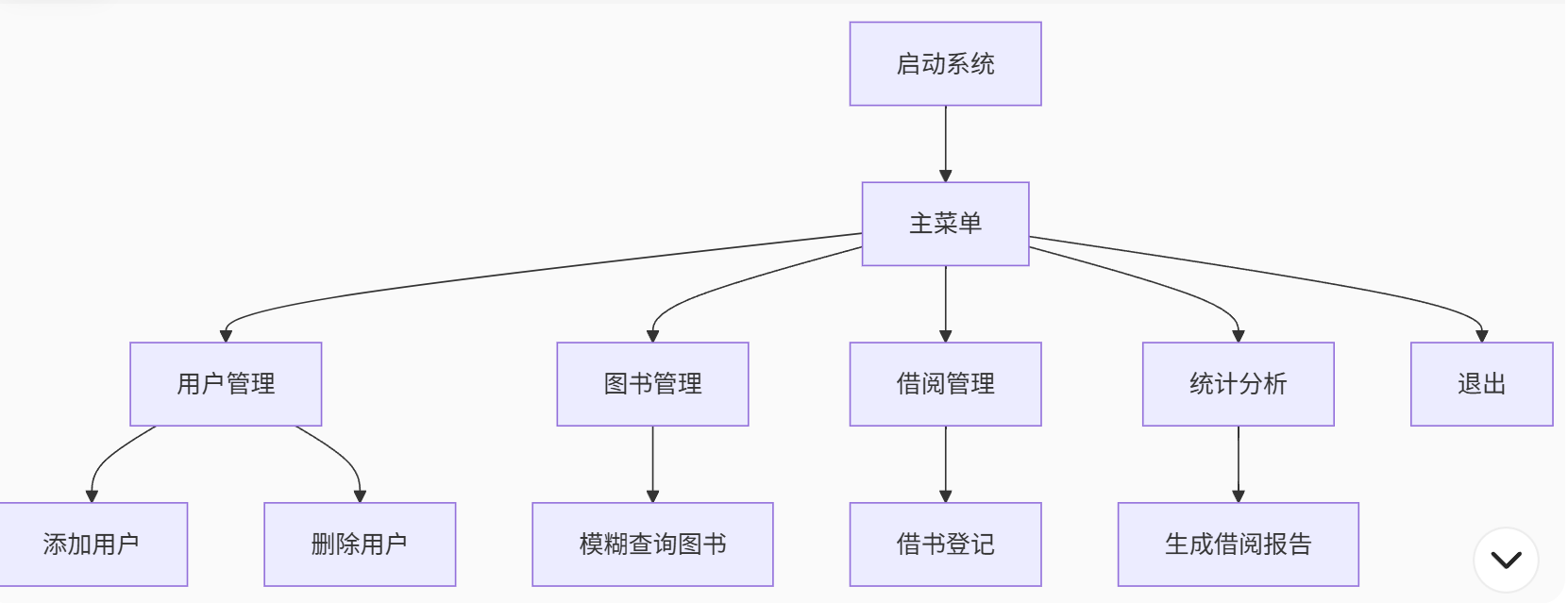
整个图书馆系统的使用从一个启动开始加载进主菜单。在主菜单中显示主要的功能模块标号12345，然后根据用于输入的标号跳转到对应的功能模块

**用户管理**：要求实现用户信息的录入、查看、修改、删除功能。用户信息包括姓名、学号、邮箱、借阅历史、违约记录。

**图书管理**：要求对每一本图书建立档案，包括图书基本信息，如书名、作者、出版年份、出版社、ISBN号码以及这本书的历史借阅记录。

**统计分析功能**：调用存储在用户系统和借阅系统两个里面的信息，生成用户自身的借阅报告和流行图书的top10排行榜（参考网易云音乐的年度听歌报告）。

**退出**：退出整个系统回到启动登录界面



3.详细设计

基本类如图所示

Record Manager

Book Borrow

Book

User

各功能模块的实现：

用户管理：

录入/删除当前用户的ID、姓名、

管理当前借阅

发送邮件提醒借阅即将逾期

提供用户信息显示接口

图书管理：

录入/删除当前图书的书名、作者、ISBN、ID

设定本书的状态：在架、借阅中、逾期未还等等

提供书籍信息显示接

借阅系统管理：

创建借阅包括借书人ID，书本ID，本次借阅的ID，借还日期，逾期提醒

1. **User 类**：
   * **属性**：
     + **userId**：用户唯一标识
     + **name**：用户姓名
     + **borrowedBooks**：当前借阅的书籍ID列表
   * **方法**：
     + **borrowBook()**：借阅书籍
     + **returnBook()**：归还书籍
     + **displayUserInfo()**：显示用户信息
2. **Book 类**：
   * **属性**：
     + **bookId**：书籍唯一标识
     + **title**：书名
     + **author**：作者
     + **isAvailable**：是否可借
     + **borrowCount**：被借阅次数
   * **方法**：
     + **updateStatus()**：更新借阅状态
     + **matchKeyword()**：实现模糊查询
     + **displayBookInfo()**：显示书籍信息
3. **BookBorrow 类**：
   * **属性**：
     + **borrowId**：借阅记录唯一标识
     + **userId**：借阅用户ID
     + **bookId**：借阅书籍ID
     + **borrowDate**：借出日期
     + **returnDate**：归还日期
   * **方法**：
     + **calculateOverdue()**：计算是否逾期
     + **displayRecord()**：显示借阅记录
4. **RecordManager 类**：
   * **属性**：
     + **allRecords**：存储所有借阅记录
   * **方法**：
     + **addRecord()**：添加新借阅记录
     + **closeRecord()**：关闭借阅记录（归还时）
     + **getUserHistory()**：获取用户借阅历史
     + **getBookHistory()**：获取书籍借阅历史
     + **findOverdueRecords()**：查找逾期记录
     + **generateStatistics()**：生成统计报告
5. 系统调试：

在初步完成编写后我进行了调试，发现自己的程序的环境配置出了问题，比如程序无法正常编译，exit -1等等，在AI的指导下，我逐步排查问题，发现是没有正确配置VS Code的编译软件，设置好了环境变量之后，这个问题得到了解决。

正式进入自己写的Library System后，我发现首页设置为用户管理、图书管理、借阅管理这几个层次不够合理，于是我做了进一步调整，先引导用户选择身份，比如管理员（Administrator）或者是普通用户（User），然后输入自己的管理员ID或者是用户ID。

对于管理员ID，将屏幕跳转为Administrator Management Page 其中包含User Management, Book Management, Record Management(管理借阅记录)。

对于User ID，将屏幕跳转为User Page，其中包含Book Search Book Borrow, Book Return等功能。

附录：源程序清单

#include<string.h>

#include<iostream>

#include<chrono>

#include"BookManager.h"

#include"UserManager.h"

#include"RecordManager.h"

#include"BookBorrow.h"

using namespace std;

void welcomePage() {

    cout << "Welcome to the Library Management System!" << endl;

    cout<<"1.Administrator"<<endl;

    cout<<"2.User"<<endl;

    cout << "3.Exit" << endl;

    cout<<"Please select your choice (1-3): "<<endl;

}

void exitPage() {

    cout << "Exiting the system. Goodbye!" << endl;

}

int main(){

    BookManager bookManager;

    UserManager userManager;

    RecordManager recordManager;

    while(1){

        welcomePage();

        int choice;

        cin >> choice;

        switch (choice) {

            case 1: {

                cout << "Administrator Mode" << endl;

                cout << "1. Book Management" << endl;

                cout << "2. User Management" << endl;

                cout << "3. Record Management" << endl;

                cout << "4. Return to Main Menu" << endl;

                cout << "Please select an option (1-4): ";

                int adminChoice;

                cin >> adminChoice;

                switch (adminChoice) {

                    case 1:

                        bookManager.bookManagementPage();

                        break;

                    case 2:

                        userManager.UserManagementPage();

                        break;

                    case 3:

                        recordManager.recordManagementPage();

                        break;

                    case 4:

                        continue; // 返回主菜单

                    default:

                        cout << "Invalid option. Please try again." << endl;

                }

                break;

            }

            case 2: {

                cout << "User Mode" << endl;

                cout << "1. Book Search" << endl;

                cout << "2. Borrow Book" << endl;

                cout << "3. Return Book" << endl;

                cout << "4. View Borrow History" << endl;

                cout << "5. Return to Main Menu" << endl;

                cout << "Please select an option (1-5): ";

                int userChoice;

                cin >> userChoice;

                switch (userChoice) {

                    case 1:

                        bookManager.bookSearchPage();

                        break;

                    case 2: {

                        int bookID, userID;

                        cout << "Enter Book ID to borrow: ";

                        cin >> bookID;

                        cout << "Enter User ID: ";

                        cin >> userID;

                        recordManager.addRecord(BookBorrow(bookID, userID));

                        cout << "Book borrowed successfully!" << endl;

                        break;

                    }

                    case 3: {

                        int bookID, userID;

                        cout << "Enter Book ID to return: ";

                        cin >> bookID;

                        cout << "Enter User ID: ";

                        cin >> userID;

                        // 假设有一个方法来处理归还逻辑

                        recordManager.returnBook(bookID, userID);

                        cout << "Book returned successfully!" << endl;

                        break;

                    }

                    case 4:

                        recordManager.listRecords();

                        break;

                    case 5:

                        continue; // 返回主菜单

                    default:

                        cout << "Invalid option. Please try again." << endl;

                }

                break;

            }

            case 3:

                cout << "Exiting the system. Goodbye!" << endl;

                return 0;

            default:

                cout << "Invalid option. Please try again." << endl;

        }

    }

}

#include<iostream>

#include<string>

#pragma once

using namespace std;

class Book {

private:

    int bookID;

    string title;

    string author;

    bool isAvailable;

    int borrowedcount;

public:

    Book(int id=00000000, string t="Unknown", string a="Unknown") : bookID(id), title(t), author(a), isAvailable(true), borrowedcount(0) {}

    inline void updateStatus(bool status) {

        isAvailable = status;

    }

    inline void displayBookInfo() const {

        cout << "Book ID: " << bookID << ", Title: " << title << ", Author: " << author;

        cout << ", Available: " << (isAvailable ? "Yes" : "No") << endl;

    }

    inline int getBookID() const {

        return bookID;

    }

    inline void setInfo(const string& t, const string& a) {

        title = t;

        author = a;

    }

    inline string getTitle() const {

        return title;

    }

    inline string getAuthor() const {

        return author;

    }

};

#include <iostream>

#include <string>

#include "Book.h"

#include "User.h"

#include <sstream>

#include <iomanip>

#include <chrono>

using namespace std;

#pragma once

string generateBorrowID(int &bookID, int &userID) {

    std::ostringstream oss;

    oss << std::setw(4) << std::setfill('0') << bookID

        << std::setw(4) << std::setfill('0') << userID;

    return oss.str();

}

long CurrentDate() {

    auto now = chrono::system\_clock::now();

    time\_t now\_c = chrono::system\_clock::to\_time\_t(now);

    cout << "Current date and time: " << ctime(&now\_c);

    return static\_cast<int>(now\_c);

}

long DateCauculate(string Date) {

    // Assuming Date is in the format "YYYY-MM-DD"

    int year, month, day;

    sscanf(Date.c\_str(), "%d-%d-%d", &year, &month, &day);

    return (year)\* 365 + month \* 30 + day; // Convert to YYYYMMDD format

}

class BookBorrow: public Book, public User {

private:

    static int BorrowDuration;

    string borrowID;

    string borrowDate;

    string returnDate;

public:

    BookBorrow(int bID=0,int uID=0, string bDate=""){

        borrowID= generateBorrowID(bID, uID);

        borrowDate=bDate;

    }

    bool CauculateOverdue() {

        if(CurrentDate() - DateCauculate(borrowDate) > BorrowDuration) {

            return true;

        }

        return false;

    }

    void displayBorrowInfo() const {

        cout << "Borrow ID: " << borrowID << ", Borrow Date: " << borrowDate;

        cout << ", Return Date: " << returnDate << endl;

        Book::displayBookInfo();

        User::displayUserInfo();

    }

};

int BookBorrow::BorrowDuration = 100;

#include <vector>

#include <algorithm>

#include <string>

#include "Book.h"

class BookManager {

private:

    vector<Book> books;

public:

    void addBook(const Book& book) { books.push\_back(book); }

    bool deleteBook(const int& bookID) {

        auto it = remove\_if(books.begin(), books.end(), [&](const Book& b){ return b.getBookID() == bookID; });

        if (it != books.end()) {

            books.erase(it, books.end());

            return true;

        }

        return false;

    }

    Book\* findBook(const int & bookID) {

        for (auto& book : books) {

            if (book.getBookID() == bookID) return &book;

        }

        return nullptr;

    }

    void modifyBook(const int& bookID, const string& title, const string& author) {

        Book\* book = findBook(bookID);

        if (book) book->setInfo(title, author);

    }

    // 模糊查询

    void fuzzySearch(const string& keyword) {

        for (const auto& book : books) {

            if (book.getTitle().find(keyword) != string::npos ||

                book.getAuthor().find(keyword) != string::npos) {

                book.displayBookInfo();

            }

        }

    }

    void listBooks() {

        for (const auto& book : books) book.displayBookInfo();

    }

    void bookSearchPage() {

        cout << "Book Search Page" << endl;

        cout << "1. Search by Title" << endl;

        cout << "2. Search by Author" << endl;

        cout << "3. Return to Main Menu" << endl;

        cout << "Please select an option (1-3): ";

        int choice;

        cin >> choice;

        switch(choice) {

            case 1: {

                string title;

                cout << "Enter Title to search: ";

                cin.ignore();

                getline(cin, title);

                fuzzySearch(title);

                break;

            }

            case 2: {

                string author;

                cout << "Enter Author to search: ";

                cin.ignore();

                getline(cin, author);

                fuzzySearch(author);

                break;

            }

            case 3:

                return; // 返回主菜单

            default:

                cout << "Invalid option. Please try again." << endl;

        }

    }

    void bookManagementPage() {

        cout << "Book Management Page" << endl;

        cout << "1. Add Book" << endl;

        cout << "2. Delete Book" << endl;

        cout << "3. Modify Book" << endl;

        cout << "4. List Books" << endl;

        cout << "5. Return to Main Menu" << endl;

        cout << "Please select an option (1-5): ";

        int choice;

        cin >> choice;

        switch(choice){

            case 1: {

                int id;

                string title, author;

                cout << "Enter Book ID: ";

                cin >> id;

                cout << "Enter Title: ";

                cin.ignore();

                getline(cin, title);

                cout << "Enter Author: ";

                getline(cin, author);

                addBook(Book(id, title, author));

                break;

            }

            case 2: {

                int id;

                cout << "Enter Book ID to delete: ";

                cin >> id;

                if (deleteBook(id)) {

                    cout << "Book deleted successfully." << endl;

                } else {

                    cout << "Book not found." << endl;

                }

                break;

            }

            case 3: {

                int id;

                string title, author;

                cout << "Enter Book ID to modify: ";

                cin >> id;

                cout << "Enter new Title: ";

                cin.ignore();

                getline(cin, title);

                cout << "Enter new Author: ";

                getline(cin, author);

                modifyBook(id, title, author);

                break;

            }

            case 4:

                listBooks();

                break;

            case 5:

                return; // 返回主菜单

            default:

                cout << "Invalid option. Please try again." << endl;

        }

    }

};

#include <vector>

#include <string>

#include "Book.h"

#include "User.h"

using namespace std;

#include "BookBorrow.h"

#pragma once

class RecordManager {

private:

    vector<BookBorrow> records;

public:

    void addRecord(const BookBorrow& record) { records.push\_back(record); }

    void listRecords() {

        for (const auto& rec : records) {

            // 假设BookBorrow有display方法

            rec.displayBorrowInfo();

        }

    }

    // 查询某用户的借阅历史

    void userHistory(const int& userID) {

        for (const auto& rec : records) {

            if (rec.getUserID() == userID) rec.displayBorrowInfo();

        }

    }

    void recordManagementPage() {

        cout << "Record Management Page" << endl;

        cout << "1. Add Record" << endl;

        cout << "2. List Records" << endl;

        cout << "3. User History" << endl;

        cout << "4. Book History" << endl;

        cout << "5. Return to Main Menu" << endl;

        cout << "Please select an option (1-5): ";

    }

    // 查询某图书的借阅历史

    void bookHistory(const int& bookID) {

        for (const auto& rec : records) {

            if (rec.getBookID() == bookID) rec.displayBorrowInfo();

        }

    }

    void returnBook(int bookID, int userID) {

        for (auto it = records.begin(); it != records.end(); ++it) {

            if (it->getBookID() == bookID && it->getUserID() == userID) {

                // 假设BookBorrow有一个方法来处理归还逻辑

                it->returnBook(bookID);

                records.erase(it);

                cout << "Book returned successfully!" << endl;

                return;

            }

        }

        cout << "Record not found." << endl;

    }

};

#include<string.h>

#include<iostream>

using namespace std;

#pragma once

class User{

private:

    int userID;

    string name;

    string emailaddress;

    int borrowedBooksID[10];

    int borrowedCount;

public:

    User(int id=00000000, string n="Unknown", string email="Unknown") : userID(id), name(n), emailaddress(email), borrowedCount(0) {

        memset(borrowedBooksID, 0, sizeof(borrowedBooksID));

    }

    void borrowBook(int bookID) {

        if (borrowedCount < 10) {

            borrowedBooksID[borrowedCount++] = bookID;

        } else {

            cout << "Cannot borrow more than 10 books." << endl;

        }

    }

    int getUserID() const {

        return userID;

    }

    void returnBook(int bookID) {

        for (int i = 0; i < borrowedCount; i++) {

            if (borrowedBooksID[i] == bookID) {

                borrowedBooksID[i] = borrowedBooksID[--borrowedCount];

                return;

            }

        }

        cout << "Book not found in borrowed list." << endl;

    }

    void setInfo(const string& n, const string& email) {

        name = n;

        emailaddress = email;

    }

    void displayUserInfo() const {

        cout << "User ID: " << userID << ", Name: " << name << ", Email: " << emailaddress << endl;

        cout << "Borrowed Books: ";

        for (int i = 0; i < borrowedCount; i++) {

            cout << borrowedBooksID[i] << " ";

        }

        cout << endl;

    }

};

#include <vector>

#include <algorithm>

#include"User.h"

#pragma once

class UserManager {

private:

    vector<User> users;

public:

    void addUser(const User& user) {

        users.push\_back(user);

    }

    bool deleteUser(const int& userID) {

        auto it = remove\_if(users.begin(), users.end(), [&](const User& u){ return u.getUserID() == userID; });

        if (it != users.end()) {

            users.erase(it, users.end());

            return true;

        }

        return false;

    }

    User\* findUser(const int& userID) {

        for (auto& user : users) {

            if (user.getUserID() == userID) return &user;

        }

        return nullptr;

    }

    void modifyUser(const int& userID, const string& name, const string& email) {

        User\* user = findUser(userID);

        if (user) user->setInfo(name, email);

    }

    void listUsers() {

        for (const auto& user : users) user.displayUserInfo();

    }

    void UserManagementPage() {

        cout << "User Management Page" << endl;

        cout << "1. Add User" << endl;

        cout << "2. Delete User" << endl;

        cout << "3. Modify User" << endl;

        cout << "4. List Users" << endl;

        cout << "5. Return to Main Menu" << endl;

        cout << "Please select an option (1-5): ";

    }

};

附录：评分表

|  |  |
| --- | --- |
| 项目 | 评价 |
| 设计方案的合理性与创新性 |  |
| 设计与调试结果 |  |
| 设计说明书的质量 |  |
| 程序基本要求涵盖情况 |  |
| 程序代码编写素养情况 |  |
| 课程设计周表现情况 |  |
| 综合成绩 |  |