



**数据结构课程设计报告**

**课程名称 数据结构**

**题目名称 图书信息管理系统**

**学生学院 计算机学院**

**专业班级 软件工程（卓越工程师班）**

**学 号**

**学生姓名**

**指导教师 苏 庆**

**2024 年 12 月 20 日**

目录

[一、 需求分析 3](#_Toc185803282)

[二、 概要设计 3](#_Toc185803283)

[数据类型 3](#_Toc185803284)

[模块调用关系图 5](#_Toc185803285)

[三、详细设计 6](#_Toc185803286)

[通用链表及相关操作 6](#_Toc185803287)

[通用链队列及相关操作 8](#_Toc185803288)

[Library类相关操作 9](#_Toc185803289)

[登录界面函数调用关系图 21](#_Toc185803290)

[用户管理界面函数调用关系图 21](#_Toc185803291)

[图书管理界面函数调用关系图 22](#_Toc185803292)

[借阅记录界面函数调用关系图 22](#_Toc185803293)

[四、调试分析 23](#_Toc185803294)

[开发中所遇问题及回顾分析 23](#_Toc185803295)

[算法时空分析 23](#_Toc185803296)

[经验和体会 23](#_Toc185803297)

[五、用户使用说明 23](#_Toc185803298)

[运行方式 23](#_Toc185803299)

[登录界面 24](#_Toc185803300)

[用户管理界面 24](#_Toc185803301)

[图书管理界面 26](#_Toc185803302)

[借阅记录界面 27](#_Toc185803303)

[六、测试数据及测试结果 28](#_Toc185803304)

[登录界面测试 28](#_Toc185803305)

[用户管理界面测试 28](#_Toc185803306)

[图书管理界面测试 30](#_Toc185803307)

[逾期记录界面测试 33](#_Toc185803308)

[七、附录 34](#_Toc185803309)

[程序清单 34](#_Toc185803310)

1. 需求分析
2. **课程设计目标**

使用线性数据结构实现一个图书信息管理系统，对用户信息、图书信息等进行增加、删除、更改、查询等管理，同时实现预约、逾期罚款等功能。用户信息和图书信息等使用文件来存储。

1. **输入输出形式**

输入和输出均采用字符串string的形式

1. **程序功能**

用户信息管理、图书信息管理、用户借阅和归还图书、逾期罚款、预约等功能

1. **测试数据**

用户数据自行输入、图书数据使用python爬虫在网站爬取后录入到程序中。

1. 概要设计

数据类型

**通用链表节点**

template<class T> struct Node {

    T elem;   // 链表元素

    Node<T> \*prev; // 前向指针

    Node<T> \*next; // 后向指针

    friend ostream &operator <<(ostream &output, const Node &node) {// 传入节点，重载输出流

        output << node.elem;

        return output;

    }

    friend ostream &operator <<(ostream &output, const Node \*&node) {// 传入节点指针重载输出流

        output << node->elem;

        return output;

    }

};

template<class T> class List {

private:

Node<T> \*head;

};

**通用链队列节点**

template<class Q> struct QNode {

    Q elem;            // 队列元素

    QNode<Q> \*next;

    friend ostream &operator <<(ostream &output, const QNode &qnode) {// 重载输出流

        output << qnode.elem;

        return output;

    }

    friend ostream &operator <<(ostream &output, const QNode \*&qnode) {//

        output << qnode->elem;

        return output;

    }

};

template<class Q> class Queue {

private:

    QNode<Q> \*front;//队列头指针

    QNode<Q> \*rear;//队列尾指针

};

**用户信息类**

class UserInfo {

public:

    string ID;         // 用户id

    string password;    // 密码

    int type;      // 用户类型

    double fine=0;     // 罚款

    List<Node<BookInfo>\*> books; // 已借阅的书籍

    List<Node<BookInfo>\*> reserveBooks; //预约的书籍

    List<string> booksISBN;//记录借阅的书籍ISBN，存储在文件中

};

**图书信息类**

class BookInfo {

public:

    string name;     // 书名

    string isbn;     // ISBN

    int quantity;     // 数量

    double price;     // 价格

    List<Node<UserInfo>\*> readers; // 借阅该书的读者

    Queue<QNode<UserInfo>\*> reserveReaders;    //预约队列,当可借阅书数量为0时启用

    List<string> readersID;//借阅者的id，存储在文件中

};

**预约信息类**

class ReserveInfo {

    //存储预约信息

public:

    string readerID;

    string bookISBN;

};

**借阅信息类**

class RecordInfo{

    //存储借阅信息,用来实现逾期罚款功能

public:

    string readerID;//借阅者id

    string bookISBN;//书籍isbn

    QDate returnTime;//归还时间

};

**Library类**

class Library {

public:

    List<BookInfo> books;// 图书信息链表

    List<UserInfo> users;// 用户信息链表

    List<ReserveInfo> reserves;//预约信息链表

    List<RecordInfo> records;//借阅信息链表

    QDate currentTime;//当前日期

    int days=30;//借阅期限为30天

    const char \*bookPath;//图书信息文件路径

    const char \*userPath;//用户信息文件路径

    const char \*reservePath;//预约信息文件路径

    const char \*recordPath;//借阅信息文件路径

    char DIVIDE\_CHAR;//分隔符

};

图示

描述已自动生成模块调用关系图

三、详细设计

通用链表及相关操作

图示

描述已自动生成// 获得链表长度

    int size() {

        int ret = 0;

        for (Node<T> \*p = head->next; p != head; p = p->next)

            ret++;

        return ret;

}

图示

描述已自动生成// 判断链表是否为空，空则返回true

    bool isEmpty() {

        return head->next == head;

    }

    // 根据索引值获取节点指针，若获取失败返回空指针

    Node<T>\* getNode(int idx) {

        Node<T> \*ret = nullptr;

        int len = size();

        if (idx < 0 || idx > len) {

            cerr << "无效的索引值。" << endl;

            return nullptr;

        }

        int i = 1;

        for (Node<T> \*p = head->next; p != head; p = p->next, i++) {

图示

描述已自动生成            if (i == idx) {

                ret = p;

                break;

            }

        }

        return ret;

    }

// 在指定节点后插入节点，返回插入的元素的指针，若插入失败返回空指针

    Node<T>\* add(T val, Node<T> \*pos) {

        Node<T> \*item = new Node<T>;

        if (!item) {

            cerr << "内存分配失败.请检查剩余内存是否充足。" << endl;

            return nullptr;

        }

        item->elem   = val; item->next   = pos->next;

        pos->next   = item; item->next->prev = item;

        item->prev   = pos;

        return item;

    }

图示

描述已自动生成// 删除指定位置的节点，返回被删除节点后一个节点的指针，若删除失败返回空指针

    Node<T>\* del(Node<T> \*pos) {

        if (pos == nullptr) {

            return nullptr;

        }

        if (pos == head) {

            cerr << "不能删除链表头元素。" << endl;

            return nullptr;

        }

        pos->prev->next = pos->next;

        pos->next->prev = pos->prev;

        Node<T> \*ret = pos->next;

        delete pos;

        return ret;

    }

图示

描述已自动生成// 修改链表元素，若修改成功返回改节点指针，若修改失败则返回空指针

    Node<T>\* modify(Node<T> \*pos, T val) {

        if (pos == nullptr) {

            return nullptr;

        }

        if (pos == head) {

            cerr << "不能修改链表头元素。" << endl;

            return nullptr;

        }

        pos->elem = val;

        return pos;

图示

描述已自动生成    }

// 查找第一个值为val的节点

    Node<T>\* find(T val) {

        for (Node<T> \*p = head->next; p != head; p = p->next) {

            if (p->elem == val) return p;

        }

        return nullptr;

    }

// 清空并重置链表

    int clear() {

        for (Node<T> \*p = head->prev->prev; p != head->prev; p = p->prev)

            delete p->next;

        head->prev = head;

        head->next = head;

        return 0;

    }

图示

描述已自动生成// 返回链表第一个节点的指针

    Node<T>\* begin() {

        return head->next;

    }

// 返回链表最后一个节点的后一个元素的指针（即头节点的指针）

    Node<T>\* end() {

        return head;

    }

 // 在链表末尾插入元素

    Node<T>\* append(T val) {

        return add(val, head->prev);

    }

图示

描述已自动生成

通用链队列及相关操作

// 初始化队列

    Queue() {

        front = nullptr;

        rear = nullptr;

    }

int size() {

        // 计算队列长度

        if (rear==nullptr||front==nullptr) return 0;

        int count = 0;

        for (QNode<Q> \*p = front; p != nullptr; p = p->next)

            count++;

        return count;

    }

图示

描述已自动生成bool enqueue(Q &val) {

        // 入队操作

        QNode<Q> \*item = new QNode<Q>;

        if (!item) {

            cerr << "内存分配失败.请检查剩余内存是否充足。" << endl;

            return false;

        }

        item->elem = val;

        item->next = nullptr;

        if (front == nullptr) {

            front = rear = item;

        } else {

            rear->next = item; rear = item;

        }

        return true;}

图示

描述已自动生成bool dequeue(Q& val) {

        // 出队操作

        if (front == nullptr) {

            cerr << "队列为空。" << endl;

            return false;

        }

        QNode<Q> \*item = front;

        val = item->elem;

        front = front->next;

        if (front == nullptr) {

            rear = nullptr;

        }

        delete item;

        return true;

    }

图示

描述已自动生成QNode<Q>\* getFront() {

图示

描述已自动生成        // 返回队首元素

        if (front == nullptr) {

            cerr << "队列为空。" << endl;

            return nullptr;

        }

        return front;

    }

    void clear() {

        // 清空队列

        while (front == nullptr) {

            Q temp;

            dequeue(temp);

        }

    }

Library类相关操作

// 从文件读取数据

    int read(const char \*userFile, const char \*bookFile,const char \*reserveFile,const char \*recordFile) {

        //读取成功返回0，失败返回1

        bookPath = bookFile;

        userPath = userFile;

        reservePath = reserveFile;

        recordPath = recordFile;

        int userState = userDataReader(userFile);

        int bookState = bookDataReader(bookFile);

        int reserveState = reserveDataReader(reserveFile);

        int recordState = recordDataReader(recordFile);

        if (userState || bookState|| reserveState||recordState) {

            cerr << "未读取到数据。" << endl;

            return 1;

        }

        // 预处理编号数据，添加节点指针到链表中

        for (auto \*p = books.begin(); p != books.end(); p = p->next) {

            // 遍历图书链表，添加节点指针到bookInfo读者链表

            auto readersID = p->elem.readersID;

            for (auto \*q = readersID.begin(); q != readersID.end(); q = q->next) {

                p->elem.readers.append(findUser(q->elem));

            }

            readersID.clear();

        }

        for (auto \*p = users.begin(); p != users.end(); p = p->next) {

            // 遍历用户链表，添加节点指针到userInfo书籍链表

            auto booksID = p->elem.booksISBN;

            for (auto \*q = booksID.begin(); q != booksID.end(); q = q->next) {

                p->elem.books.append(findBookbyISBN(q->elem));

            }

            booksID.clear();

        }

        for(auto \*p = reserves.begin(); p!= reserves.end(); p = p->next){

图示

描述已自动生成            //遍历预约链表，添加节点指针到reserveInfo读者链表

            auto readerID = p->elem.readerID;

            auto bookISBN = p->elem.bookISBN;

            auto reserveUser = findUser(readerID);

            QNode<UserInfo> \*reserveNode = new QNode<UserInfo>;

            reserveNode->elem = reserveUser->elem;//转换为QNode类型

            auto reserveBook = findBookbyISBN(bookISBN);

            if(reserveUser && reserveBook){

                //信息都存在

                reserveUser->elem.reserveBooks.append(reserveBook);

                reserveBook->elem.reserveReaders.enqueue(reserveNode);

            }

        }

        for(auto \*p=records.begin();p!=records.end();p=p->next){

            //遍历借阅记录链表，将逾期未归还的记录添加到用户罚款中

            if(p->elem.returnTime<currentTime){

                //逾期

                auto user = findUser(p->elem.readerID);

                auto book = findBookbyISBN(p->elem.bookISBN);

                if(user && book){

                    user->elem.fine += book->elem.price;

                } else{

                    cerr << "借阅记录中读者或书籍不存在。" << endl;

图示

描述已自动生成                }

            }

        }

        return 0;

    }

// 写入文件信息

    int writeBook(const char \*bookFile) {

        ofstream output(bookFile);

        if (!output) {

            cerr << "无法写入文件。请检查文件\"" << bookFile << "\"是否被占用。" << endl;

            return 1;

        }

        for (auto \*p = books.begin(); p != books.end(); p = p->next) {

            output << p->elem.name << DIVIDE\_CHAR << p->elem.isbn << DIVIDE\_CHAR

                   << p->elem.quantity<< DIVIDE\_CHAR << p->elem.price;

            auto readers = p->elem.readers;

            for (auto \*q = readers.begin(); q != readers.end(); q = q->next) {

图示

描述已自动生成                Node<UserInfo> \*user = q->elem;

                output << DIVIDE\_CHAR << user->elem.ID;

            }

            output << endl;

        }

        return 0;

    }

    int writeUser(const char \*userFile) {

        ofstream output(userFile);

        if (!output) {

            cerr << "无法写入文件。请检查文件\"" << userFile << "\"是否被占用。" << endl;

            return 1;

        }

        for (auto \*p = users.begin(); p != users.end(); p = p->next) {

            output << p->elem.ID<< DIVIDE\_CHAR << p->elem.password

                   <<  DIVIDE\_CHAR << p->elem.type<< DIVIDE\_CHAR <<p->elem.fine;

            auto books = p->elem.books;

            for (auto \*q = books.begin(); q != books.end(); q = q->next) {

                Node<BookInfo> \*book = q->elem;

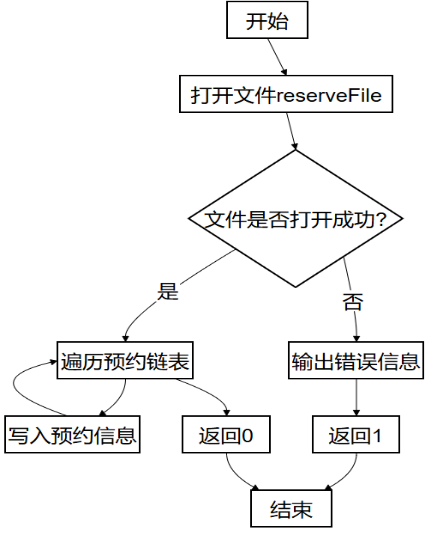
                output << DIVIDE\_CHAR << book->elem.isbn;

            }

            output << endl;

        }

        return 0;

    }

    int writeReserve(const char \*reserveFile){

        //写入预约信息

        ofstream output(reserveFile);

        if (!output) {

            cerr << "无法写入文件。请检查文件\"" << reserveFile << "\"是否被占用。" << endl;

            return 1;

        }

        for(auto \*p = reserves.begin(); p!= reserves.end(); p = p->next){

图示

描述已自动生成            output << p->elem.readerID<< DIVIDE\_CHAR << p->elem.bookISBN << endl;

        }

        return 0;

    }

    int writeRecord(const char \*recordFile){

        //写入借阅信息

        ofstream output(recordFile);

        if (!output) {

            cerr << "无法写入文件。请检查文件\"" << recordFile << "\"是否被占用。" << endl;

            return 1;

        }

        for(auto \*p = records.begin(); p!= records.end(); p = p->next){

            output << p->elem.readerID << DIVIDE\_CHAR << p->elem.bookISBN << DIVIDE\_CHAR << p->elem.returnTime.toString("yyyy-MM-dd").toStdString() << endl;

        }

        return 0;

    }

// 按编号查找图书

    Node<BookInfo>\* findBookbyISBN(string isbn) {

        for (auto \*p = books.begin(); p != books.end(); p = p->next) {

            //遍历图书链表，直到找到目标图书

            if (p->elem.isbn == isbn) return p;

        }

        return nullptr;

    }

    // 按id查找用户

    Node<UserInfo>\* findUser(string id) {

        for (auto \*p = users.begin(); p != users.end(); p = p->next) {

            //遍历用户链表，直到找到目标用户

            if (p->elem.ID == id) return p;

        }

        return nullptr;

    }

    // 按名称查找图书

    Node<BookInfo>\* findBookbyName(string name) {

        for (auto \*p = books.begin(); p != books.end(); p = p->next) {

            //遍历图书链表，直到找到目标图书

            if (p->elem.name == name) return p;

        }

        return nullptr;

    }

    //查找借阅记录

    Node<RecordInfo>\* findRecord(string readerID, string bookISBN){

        for(auto \*p = records.begin(); p!= records.end(); p = p->next){

            //遍历借阅记录链表，直到找到目标记录

            if(p->elem.readerID == readerID && p->elem.bookISBN == bookISBN){

                return p;}

            }

        return nullptr;

    }

    // 按名称查找图书（模糊查找），返回一个链表，存有目标图书的节点指针

图示

描述已自动生成    List<Node<BookInfo>\*> fuzzyFindBook(string name) {

        List<Node<BookInfo>\*> ret;

        for (auto \*p = books.begin(); p != books.end(); p = p->next) {

            //遍历图书链表，查找名称包含name的图书

            if (p->elem.name.find(name) != string::npos) {

                ret.append(p);

            }

        }

        return ret;

}

// 添加图书信息

Node<BookInfo>\* add(BookInfo book) {

        return books.append(book);

    }

    // 添加用户信息

    Node<UserInfo>\* add(UserInfo user) {

        return users.append(user);

    }

    // 添加借阅信息

    Node<RecordInfo>\* add(RecordInfo record){

        return records.append(record);

    }

Node<ReserveInfo>\* addReserve(ReserveInfo reserve) {

        //增加预约信息，同时更新用户链表和图书链表中的预约信息

        auto book = findBookbyISBN(reserve.bookISBN);

        auto user = findUser(reserve.readerID);

        if(!book || !user) return nullptr;

图示

描述已自动生成        if(book->elem.quantity>(book->elem.readers.size()+book->elem.reserveReaders.size())){

            return nullptr;//数量足够，不需要预约

        }

        auto data = new QNode<UserInfo>;

        data->elem = user->elem;

        book->elem.reserveReaders.enqueue(data);//进队

        user->elem.reserveBooks.append(book);

        return reserves.append(reserve);

    }

    Node<ReserveInfo>\* delReserve(ReserveInfo reserve){

        //删除预约信息，同时更新用户链表和图书链表中的预约信息

        auto book = findBookbyISBN(reserve.bookISBN);

        auto user = findUser(reserve.readerID);

        if(!book || !user) return nullptr;

        auto data = book->elem.reserveReaders.getFront()->elem;

        if(data->elem.ID!= reserve.readerID) return nullptr;//不是最先的预约者

        QNode<UserInfo> \*node;

图示

描述已自动生成        book->elem.reserveReaders.dequeue(node);

        user->elem.reserveBooks.delByValue(book);

        reserves.delByValue(reserve);

        Node<ReserveInfo>\* p = new Node<ReserveInfo>;

        p->elem = reserve;

        return p;

    }

// 删除图书节点，force=true 开启强制删除

    Node<BookInfo>\* del(Node<BookInfo>\* book, bool force = false) {

        if (book == nullptr) {

            cerr << "不存在符合条件的图书。" << endl;

            return nullptr;

        }

        auto readers = book->elem.readers;//获取读者链表

        if (!readers.isEmpty()) {

            cerr << "[警告] 现在还有 " << readers.size() << " 名用户未还该书 《"

                 << book->elem.name << "》(" << book->elem.isbn << ")。" << endl;

            if (!force) return nullptr;

        }

        for (auto \*p = readers.begin(); p != readers.end(); p = p->next) {

            //遍历读者链表，删除读者信息

            Node<UserInfo> \*user = p->elem;

            user->elem.books.delByValue(book);

        }

        return books.del(book);

图示

描述已自动生成    }

    // 删除用户节点，force=true 开启强制删除,并强制归还该书

    Node<UserInfo>\* del(Node<UserInfo>\* user, bool force = false) {

        if (user == nullptr) {

            cerr << "不存在符合条件的用户。" << endl;

            return nullptr;

        }

        auto books = user->elem.books;//获取书籍链表

        if (!books.isEmpty()) {

            cerr << "[警告] 该用户" << user->elem.ID

                  << "未还图书 " << books.size() << " 本。";

            if (!force) return nullptr;

        }

图示

描述已自动生成        for (auto \*p = books.begin(); p != books.end(); p = p->next) {

            //遍历书籍链表，删除书籍信息

            Node<BookInfo> \*book = p->elem;

            book->elem.readers.delByValue(user);

        }

        return users.del(user);

    }

int borrowBook(Node<UserInfo>\* userNode, Node<BookInfo>\* bookNode) {

        //借书函数  成功返回0，失败返回1,预约返回2,逾期返回3

        if (!userNode || !bookNode) {

            cerr << "不存在符合条件的图书或用户。" << endl;

            return 1;

        }

        if(userNode->elem.fine>0) return 3; //用户有罚款,不能借书

        BookInfo book = bookNode->elem;

        // 判断书是否还有剩余

        int quantity = book.quantity;

        if (quantity <= book.readers.size()) {

          // 图书数量不足

                //预约

                addReserve(ReserveInfo(userNode->elem.ID, bookNode->elem.isbn));

                return 2;

        }

        auto retUser = userNode->elem.books.append(bookNode);

        auto retBook = bookNode->elem.readers.append(userNode);

        //添加借阅记录

        auto record=add(RecordInfo(userNode->elem.ID,

                                   bookNode->elem.isbn,currentTime.addDays(days)));

        // cout << retBook->elem->elem << "借了" << retUser->elem->elem << endl;

        if (!retUser || !retBook||!record) return 1;

        return 0; }

图示

描述已自动生成int returnBook(Node<UserInfo>\* userNode, Node<BookInfo>\* bookNode) {

        if (!userNode || !bookNode) {

            cerr << "不存在符合条件的图书或用户。" << endl;

            return 1;

        }

        //删除用户信息

        auto retUser = userNode->elem.books.delByValue(bookNode);

        //删除图书信息

        auto retBook = bookNode->elem.readers.delByValue(userNode);

        //删除借阅记录

        auto retRecord= delRecord(userNode->elem.ID,bookNode->elem.isbn);

        if (!retUser || !retBook||!retRecord) return 1;//删除失败

        //还书后，判断是否有预约

        if(bookNode->elem.reserveReaders.size() > 0){

            //有预约，取出最先的预约者

            auto node = bookNode->elem.reserveReaders.getFront();

            auto userid = node->elem->elem.ID;

            //借书

            delReserve(reserves.begin()->elem);//删除预约信息

            //借书

            borrowBookByISBN(userid,bookNode->elem.isbn);

        }

        return 0;

图片包含 图示

描述已自动生成    }

// 密码验证，判断是否登录成功

    bool login(string userID, string password) {

        auto result = findUser(userID);

        if (result) {

            return result->elem.password == password;

        } else {

            return false;

        }

    }

int bookDataReader(const char \*fileName) {

        ifstream input(fileName);

        if (!input) {

            qDebug() << "数据读取失败。请检查文件\"" << fileName << "\"是否存在。";

            return 1;

        }

        while (!input.eof()) {

            string line;

            getline(input, line);  // 读入一行

            if (line.empty()) continue; // 若读到空行则跳过

            line += '\n';

            string buff;  // 临时存储块

            string name;  // 图书的名称

            int cnt = 0;  // 计算由分隔符隔开的块索引号

            string isbn;  // 图书的编号

            int quantity;  // 图书的数量

            double price;  // 图书的价格

            List<string> IDs;  // 借阅图书的用户编号

            // 处理该行数据，按分隔符分隔处理

            for (string::iterator i = line.begin(); i != line.end(); i++) {

图示

描述已自动生成                if (\*i == DIVIDE\_CHAR || \*i == '\n') {

                    switch (cnt) {

                        case 0:

                            name = buff;

                            buff.clear(); break;

                        case 1:

                            isbn = buff;

                            buff.clear(); break;

                        case 2:

                            quantity = atoi(buff.data());

                            buff.clear(); break;

                        case 3:

                            price = atof(buff.data());

                            buff.clear(); break;

                        default:

                            string id = buff.data();

                            if (!id.empty()) IDs.append(id);

                            buff.clear(); break;

                    }

                    cnt++;

                } else {

                    buff += \*i;

                }

            }

            books.append(BookInfo(name,isbn, quantity, price,IDs));

        }

        input.close();

        return 0;

    }

int userDataReader(const char \*fileName) {

        // 从文件读取用户数据

        ifstream input(fileName);

        if (!input) {

            qDebug() << "数据读取失败。请检查文件\"" << fileName << "\"是否存在。";

            return 1;

        }

        while (!input.eof()) {

            string line;

            getline(input, line);  // 读入一行

            if (line.empty()) continue; // 若读到空行则跳过

            line += '\n';

            string buff;  // 临时存储块

            string ID;      // 用户的id

            string password; // 用户的密码

            int cnt = 0;  // 计算由分隔符隔开的块索引号

            int type;  // 用户类型 0：非管理员；1：管理员

            double fine;  // 罚款金额

            List<string> ISBN;  // 用户借阅的图书编号

            // 处理该行数据，按分隔符分隔处理

            for (string::iterator i = line.begin(); i != line.end(); i++) {

图示

描述已自动生成                if (\*i == DIVIDE\_CHAR || \*i == '\n') {

                    switch (cnt) {

                        case 0:

                            ID = buff;

                            buff.clear(); break;

                        case 1:

                            password = buff;

                            buff.clear(); break;

                        case 2:

                            type = atoi(buff.data());

                            buff.clear(); break;

                        case 3:

                            fine = atof(buff.data());

                            buff.clear(); break;

                        default:

                            string isbn = buff;

                            if (!isbn.empty()) ISBN.append(isbn);

                            buff.clear(); break;

                    }

                    cnt++;

                } else {

                    buff += \*i;

                }

            }

            users.append(UserInfo(ID, password,type,fine,ISBN));

        }

        input.close();

        return 0;

    }

int reserveDataReader(const char \*fileName) {

        ifstream input(fileName);

        if (!input){

            qDebug() << "数据读取失败。请检查文件\"" << fileName << "\"是否存在。";

            return 1;

        }

        while (!input.eof()) {

            string line;

            getline(input, line);  // 读入一行

            if (line.empty()) continue; // 若读到空行则跳过

            line += '\n';

            string buff;  // 临时存储块

            string ID;      // 用户的id

图示

描述已自动生成            string bookISBN; // 预约的图书编号

            int cnt = 0;  // 计算由分隔符隔开的块索引号

            // 处理该行数据，按分隔符分隔处理

            for (string::iterator i = line.begin(); i != line.end(); i++) {

                if (\*i == DIVIDE\_CHAR || \*i == '\n') {

                    switch (cnt) {

                        case 0:

                            ID = buff;

                            buff.clear(); break;

                        default:

                            bookISBN = buff;

                            buff.clear(); break;

                    }

                    cnt++;

                } else {

                    buff += \*i;

                }

            }

            reserves.append(ReserveInfo(ID, bookISBN));

        }

        input.close();

        return 0;

    }

int recordDataReader(const char \*fileName) {

        // 从文件读取借阅记录数据

        ifstream input(fileName);

        if (!input) {

            qDebug() << "数据读取失败。请检查文件\"" << fileName << "\"是否存在。";

            return 1;

        }

        while (!input.eof()) {

            string line;

            getline(input, line);  // 读入一行

            if (line.empty()) continue; // 若读到空行则跳过

            line += '\n';

            string buff;  // 临时存储块

            string ID;      // 用户的id

            string bookISBN; // 借阅的图书编号

            string time;  // 归还时间

            int cnt = 0;  // 计算由分隔符隔开的块索引号

            // 处理该行数据，按分隔符分隔处理

图示

描述已自动生成            for (string::iterator i = line.begin(); i != line.end(); i++) {

                if (\*i ==DIVIDE\_CHAR  || \*i == '\n') {

                    switch (cnt) {

                        case 0:

                            ID = buff;

                            buff.clear(); break;

                        case 1:

                            bookISBN = buff;

                            buff.clear(); break;

                        default:

                            time = buff;

                            buff.clear(); break;

                    }

                    cnt++;

                } else {

                    buff += \*i;

                }

            }

            QString time1= QString::fromStdString(time);//转换为Qstring类型

            QDate date = QDate::fromString(time1, "yyyy-MM-dd");//可能因格式问题导致转换失败

            if(!date.isValid()){

                qDebug() << "日期格式错误，请检查文件\"" << fileName << "\"。";

            }

            records.append(RecordInfo(ID, bookISBN,date));

        }

        input.close();

        return 0;

    }

图示

描述已自动生成登录界面函数调用关系图

图示, 示意图

描述已自动生成用户管理界面函数调用关系图

图书管理界面函数调用关系图

图示, 示意图

描述已自动生成

图示

描述已自动生成借阅记录界面函数调用关系图

四、调试分析

开发中所遇问题及回顾分析

1. 在项目开发的初期阶段，由于采用跨平台开发工具（CLion和Qt Creator）进行编程，导致了在利用CMake进行编译时出现了问题。经过查阅相关文档和资料，我们发现将Qt框架从Qt5升级到Qt6后，CMake编译的问题得到了解决。
2. 在开发用户管理功能时，程序运行过程中出现了一个bug。经过仔细的调试和排查，我们发现问题是由于函数中存在一个空指针所引起的。因此，在开发过程中，应该先对每个函数进行单独测试，然后再进行整体的集成测试
3. 在编写预约功能时，程序运行后我们发现还书操作后队列无法正确出队。经过调试，我们发现这一问题是由于借书逻辑中存在错误所导致的。

算法时空分析

* **查找操作：**查找操作采用遍历链表的形式来实现，代码通过一个 for 循环遍历链表中的每个节点，直到找到目标值 val 或遍历完整个链表。时间复杂度为O(n)。如果希望进一步优化，可以考虑以下方法：**哈希表**：如果需要频繁查找，可以将链表中的元素存储到哈希表中，这样查找的时间复杂度可以降低到 **O(1)**。
* **删除操作和修改操作：**该操作采用遍历查找后修改或删除实现，时间复杂度为O(n)。
* **预约操作：**预约操作采用链队列的数据结构实现。入队和出队的时间复杂度均为O(n)。
* **罚款操作：**罚款操作和清缴罚款都是采用遍历整个链表的形式来实现，因此时间复杂度为O(n)。

经验和体会

通过本次课程设计，我对C++语言的面向对象特性、模板编程使用有了更深入的理解和实践。在开发过程中，对Qt框架的信号与槽机制、事件处理以及UI设计等方面有了更加熟练的运用。采用双向链表作为数据结构，使得图书管理系统在数据操作上更加高效，体会到了选择合适的数据结构对于系统性能的重要性。在项目开发过程中，实践了软件工程的基本原则，包括需求分析、系统设计、编码实现、测试与调试等，提升了我的软件工程实践能力。通过本次课程设计，不仅提升了我在C++语言和Qt框架方面的技术水平，也加深了我对软件工程实践流程的理解。未来，我将继续深化相关技术的学习，并将所学知识应用于更多的实际项目中。

五、用户使用说明

运行方式

软件采用clion2023.3.4和Qt creator14.0.2开发。点击cmake-build-debug文件夹内exe文件即可运行。

图形用户界面, 文本, 应用程序, 表格, 电子邮件

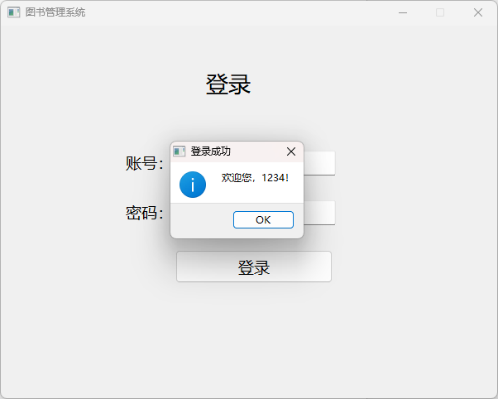
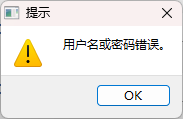
描述已自动生成图形用户界面, 应用程序, 电子邮件

描述已自动生成

图形用户界面

描述已自动生成登录界面

* 当输入用户名为空时，会弹出提示
* 初始账号为：“1234”、“1234” ，当输入的账号密码不存在时，会有提示

图形用户界面

描述已自动生成

表格

描述已自动生成用户管理界面

* 管理员能够看到和管理所有账号，普通用户只能看到当前登录的账号。

图形用户界面, 应用程序

描述已自动生成

* **搜索功能：**在输入搜索框内输入用户ID并点击查询按钮（当搜索框内为空时显示全部账号）

图形用户界面, 应用程序

描述已自动生成

* **增加用户功能：仅管理员可见**，点击后会弹出用户详细信息界面，输入用户id后可创建账号。密码需要在创建账号后更改。创建账号时可选是否为管理员账号。

图形用户界面, 应用程序

描述已自动生成

* **用户详细信息功能：**双击账号或点击详细信息按钮后可打开用户详细信息界面。

图形用户界面

描述已自动生成图形用户界面, 应用程序, Word

描述已自动生成

* 图形用户界面, 表格

  中度可信度描述已自动生成**修改账号信息功能及还书功能：（**修改信息后要点击OK按钮才能保存**）**

图形用户界面

描述已自动生成

图书管理界面

* **查询功能：**可按书名或书号查询

图形用户界面

低可信度描述已自动生成

* **借书还书及预约功能：**单击选中书籍后可使用，当选中书籍无库存时会借书自动进行预约。图书详细信息界面内的功能同理。（有罚款的用户无法借书）

图形用户界面, 应用程序

描述已自动生成 电脑软件截图

描述已自动生成

* **查看图书详细信息功能：**双击图书或单击图书后点击详细信息按钮可进入图书详细信息界面

图形用户界面, 应用程序

描述已自动生成 图形用户界面, 文本, 应用程序

描述已自动生成

* **增加图书功能：**点击增加图书按钮后即可编辑图书信息，点击ok后即可保存（仅管理员可增加）
* **修改、删除图书信息功能：**进入图书详细信息界面后，可修改图书信息，点击ok按钮即可保存（仅管理员可修改）

图形用户界面, 应用程序, Word

描述已自动生成

借阅记录界面

（该界面仅管理员可见）

* **搜索功能：**有三个搜索方式可选。同时也可选择仅查看逾期记录。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

* **日期修改功能：**直接修改日期框即可修改当前日期

图形用户界面, 文本, 应用程序

描述已自动生成图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

* **罚款清缴功能：**单击要还款的账号，再点击清缴罚款按钮

六、测试数据及测试结果

登录界面测试

图形用户界面

描述已自动生成图形用户界面, 文本

描述已自动生成图形用户界面

描述已自动生成

用户管理界面测试

* **搜索**

表格

描述已自动生成 图形用户界面, 应用程序

描述已自动生成

* **增加用户**

表格

描述已自动生成 图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

表格

描述已自动生成 图形用户界面, 文本

描述已自动生成

* **修改功能:**

文本

描述已自动生成 文本

描述已自动生成

图形用户界面, 文本

描述已自动生成

* **密码修改测试**

图形用户界面

描述已自动生成 社交网站的手机截图

描述已自动生成

* **还书测试**

表格

描述已自动生成 图形用户界面, 表格

描述已自动生成

图书管理界面测试

* **搜索测试**

图形用户界面, 表格

描述已自动生成 图形用户界面, 表格

中度可信度描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

* **增加图书测试**

表格

描述已自动生成 图形用户界面, 文本, 应用程序

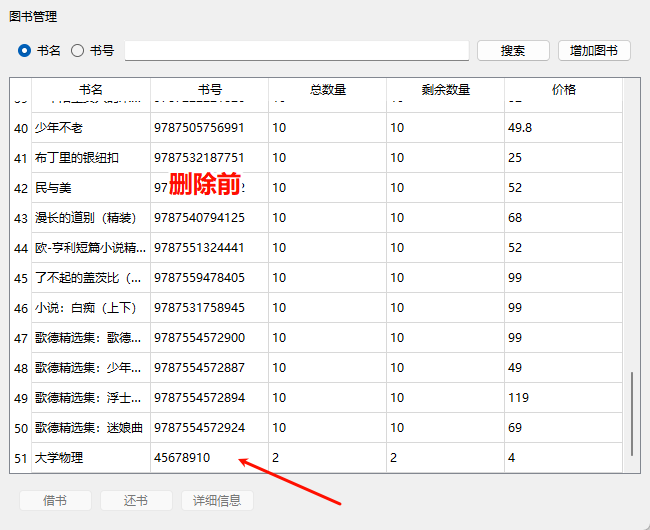
描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成 图形用户界面

描述已自动生成

* **删除图书信息测试**

 图片包含 图形用户界面

描述已自动生成

* **更改图书信息测试**

表格

描述已自动生成 图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 应用程序

描述已自动生成 表格

描述已自动生成

* **借书测试**

文本

描述已自动生成 图形用户界面, 文本, 应用程序

描述已自动生成

* **预约测试**

表格

描述已自动生成 图形用户界面, 应用程序

描述已自动生成

图形用户界面, 文本

描述已自动生成 图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面

描述已自动生成 图形用户界面, 文本, 应用程序

描述已自动生成

* **还书测试**

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成 图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

逾期记录界面测试

* **借书还书测试**

表格

描述已自动生成 图形用户界面, 表格

中度可信度描述已自动生成

表格

描述已自动生成

* **日期更改测试**

表格

描述已自动生成表格

中度可信度描述已自动生成

* **罚款缴纳测试**

图形用户界面

描述已自动生成图形用户界面

中度可信度描述已自动生成

图形用户界面, 应用程序, Word

描述已自动生成图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

* **查询测试**

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成图形用户界面

描述已自动生成

七、附录

程序清单

//Library.h

int read(const char \*userFile, const char \*bookFile,const char \*reserveFile,const char \*recordFile);

int writeBook(const char \*bookFile);

int writeUser(const char \*userFile);

int writeReserve(const char \*reserveFile);

int writeRecord(const char \*recordFile)

Node<BookInfo>\* findBookbyISBN(string isbn);

Node<UserInfo>\* findUser(string id);

Node<BookInfo>\* findBookbyName(string name);

Node<RecordInfo>\* findRecord(string readerID, string bookISBN);

 List<Node<BookInfo>\*> fuzzyFindBook(string name);

Node<BookInfo>\* add(BookInfo book);

Node<UserInfo>\* add(UserInfo user);

Node<RecordInfo>\* add(RecordInfo record);

Node<ReserveInfo>\* addReserve(ReserveInfo reserve);

Node<ReserveInfo>\* delReserve(ReserveInfo reserve);

Node<BookInfo>\* del(Node<BookInfo>\* book, bool force = false);

Node<UserInfo>\* del(Node<UserInfo>\* user, bool force = false);

Node<RecordInfo>\* delRecord(string readerID, string bookISBN);

int borrowBook(Node<UserInfo>\* userNode, Node<BookInfo>\* bookNode);

int borrowBookByISBN(string userID, string bookISBN);

int returnBook(Node<UserInfo>\* userNode, Node<BookInfo>\* bookNode);

 int returnBookbyISBN(string userID, string bookISBN);

bool isAdmin(Node<UserInfo>\* user);

 bool login(string userID, string password);

int bookDataReader(const char \*fileName);

int userDataReader(const char \*fileName);

int reserveDataReader(const char \*fileName);

int recordDataReader(const char \*fileName);

//admin.h

void initUI();//初始化界面

QStandardItemModel\* initUserModel(QStandardItemModel\* userModel); //初始化用户模型

QStandardItemModel\* initBookModel(QStandardItemModel\* bookModel);//初始化书籍模型

void displayUserData(QStandardItemModel\* userModel); //显示用户信息

void appendOneUser(QStandardItemModel\* userModel,Node<UserInfo>\* p); //添加一个用户到模型中

void displayOneUser(QStandardItemModel\* userModel,Node<UserInfo>\* p); //显示一个用户信息

void displayBookData(QStandardItemModel\* bookModel);//显示图书信息

void appendOneBook(QStandardItemModel\* bookModel,Node<BookInfo>\* p); //添加一个书籍到模型中

void displayOneBook(QStandardItemModel\* bookModel,Node<BookInfo>\* p); //显示一个书籍信息

void menu\_change(); //界面切换

private:

    Ui::admin \*ui;

    User\_manage \*m\_user\_manage;

    Book\_manage \*m\_book\_manage;

    Borrow\_record \*m\_borrow\_record;

//book\_manage.h

public:

    explicit Book\_manage(QWidget \*parent = nullptr);

    ~Book\_manage() override;

    void displayBookData();//显示图书数据

    string getSelect();//获取选中的图书书号

    void displaySingleBookData(Node<BookInfo> \*p);//显示单个图书数据

    void displayBookList(List<Node<BookInfo>\*> list);//显示多本图书

public slots:

    void on\_addBook\_btn\_clicked();//添加图书槽函数

    void on\_search\_btn\_clicked();//搜索图书槽函数

    void on\_book\_tableView\_clicked(const QModelIndex &index);//表格单击槽函数

    void on\_book\_tableView\_doubleClicked();//表格双击槽函数

    void on\_bookInfo\_btn\_clicked();//图书详细信息按钮

    void on\_borrowBook\_btn\_clicked();//借阅图书

    void on\_returnBook\_btn\_clicked();//归还图书

private:

    Ui::Book\_manage \*ui;

    QStandardItemModel \*bookModel;

    admin \*\_admin;

//bookinfo\_dialog.h

public:

    explicit bookinfo\_dialog(QWidget \*parent = nullptr,string \_bookIsbn = "");

    ~bookinfo\_dialog() override;

    void initUserTable();//初始化借阅用户表

    void appendOneUser(Node<UserInfo>\* p);//添加单个用户到模型

    void appendOneReserve(QNode<UserInfo>\* p);//添加单个预约用户到模型

    void displayUserData();//显示借阅者信息

    void updateBookInfo();//更新书籍信息

    bool checkBookInfo();//检查书籍信息是否合法

public slots:

    void  on\_buttonBox\_accepted();

    void on\_delete\_btn\_clicked();//删除按钮

    void on\_borrowBook\_btn\_clicked();//借阅按钮

    void on\_returnBook\_btn\_clicked();//归还按钮

private:

    Ui::bookinfo\_dialog \*ui;

    QStandardItemModel\* userModel;//借阅用户模型

    QStandardItemModel\* reserveModel;//预约用户模型

    Node<BookInfo>\* book;

//borrow\_record.h

public:

    explicit Borrow\_record(QWidget \*parent = nullptr);

    ~Borrow\_record() override;

    void initRecordModel();//初始化记录模型

    void displayRecord();//显示记录

    void displayOneRecord(Node<RecordInfo> \*p);//显示一条记录

    void appendOneRecord(Node<RecordInfo> \*p);//添加一条记录到模型中

    void clearFine();//清空罚款

    void updateRecord(); //时间变化后更新用户罚款

    string getSelect(); //获取选中的记录的用户id

    void displayUserList(QString query);//显示记录中某用户的所有借阅记录

    void displayBookNameList(QString query);//显示记录中某书名的所有借阅记录

    void displayBookList(QString query);//显示记录中某书号的所有借阅记录

public slots:

    void on\_tableView\_clicked(const QModelIndex &index);// 表格点击事件

    void on\_payFine\_btn\_clicked();//缴纳罚款按钮点击事件

    void on\_search\_btn\_clicked(); // 搜索按钮点击事件

private:

    Ui::Borrow\_record \*ui;

    QStandardItemModel \*recordModel;

//change\_psw\_dialog.h

public:

    explicit change\_psw\_dialog(QWidget \*parent = nullptr,Node<UserInfo> \*\_user = nullptr);

    ~change\_psw\_dialog() override;

signals:

    void changePsw(QString);//发送修改密码信号

public slots:

    void on\_buttonBox\_accepted();//确认修改密码槽函数

private:

    Ui::change\_psw\_dialog \*ui;

    Node<UserInfo>\* user;

    bool check\_psw();//检查密码是否正确

//user\_manage.h

public:

    explicit User\_manage(QWidget \*parent = nullptr);

    ~User\_manage() override;

    void displayUserData();//显示用户数据

    string getSelect();//获取选中的用户ID

    void displaySingleUserData(Node<UserInfo> \*p);//显示单个用户数据

public slots:

    void on\_addUser\_btn\_clicked();//添加账号

    void on\_search\_btn\_clicked();//搜索

    void on\_tableView\_clicked(const QModelIndex &index);//单击选中某一行

    void on\_tableView\_doubleClicked();//双击某一行

    void on\_userInfo\_btn\_clicked();//查看编辑用户信息

private:

    Ui::User\_manage \*ui;

    QStandardItemModel\* userModel;

    admin\* \_admin;

//userinfo\_dialog.h

public:

    explicit userinfo\_dialog(QWidget \*parent = nullptr,string \_userID="");

    ~userinfo\_dialog() override;

    bool check\_user();

    void updateUser();

    void change\_password(QString psw);//改变密码

    void initBookModel();//初始化书籍列表

    void displayBookTable();//显示书籍列表

    void appendOneBook(Node<BookInfo>\* book);//添加一本书到模型

    void appendOneReserve(Node<BookInfo>\* book);//添加一本预约书籍到模型

    string getSelect();//获取选中的书籍isbn

public slots:

    void buttonBox\_accepted();//确认后更新用户信息

    void delete\_btn\_clicked();//删除用户信息

    void changePsw\_btn\_clicked();//修改密码

    void on\_book\_tableView\_clicked(const QModelIndex &index);//单击书籍列表

    void on\_returnBook\_btn\_clicked();//归还书籍

private:

    Ui::userinfo\_dialog \*ui;

    QStandardItemModel \*bookModel;

    QStandardItemModel \*reserveModel;

    Node<UserInfo>\* user;