目录

[一、需求分析 3](#_Toc28468635)

[1.问题描述 3](#_Toc28468636)

[2.基本要求 3](#_Toc28468637)

[3.已完成情况 3](#_Toc28468638)

[4.程序亮点 4](#_Toc28468639)

[二、概要设计 5](#_Toc28468640)

[1. 数据类型的定义 5](#_Toc28468641)

[2.功能模块关系 5](#_Toc28468642)

[三、 详细设计 6](#_Toc28468643)

[1. 数据类型定义 6](#_Toc28468644)

[2. 函数功能具体设计 8](#_Toc28468645)

[四、 调试分析 14](#_Toc28468646)

[1. 问题分析与解决 14](#_Toc28468647)

[2. 算法分析 14](#_Toc28468648)

[3. 经验体会 14](#_Toc28468649)

[五、 用户使用说明 14](#_Toc28468650)

[1. 功能菜单 14](#_Toc28468651)

[2. 使用说明 14](#_Toc28468652)

[六、 测试结果 15](#_Toc28468653)

[1. 图书入库 15](#_Toc28468654)

[2. 删除图书 15](#_Toc28468655)

[3. 预约借书 16](#_Toc28468656)

[4. 借书 16](#_Toc28468657)

[5. 还书 16](#_Toc28468658)

[6. 查询图书 17](#_Toc28468659)

[7. 查询作者 18](#_Toc28468660)

[8. 查询读者 18](#_Toc28468661)

[9. 查看全部（打印B树） 19](#_Toc28468662)

[10. 操作日志 19](#_Toc28468663)

[七、 附录 19](#_Toc28468664)

[1. 头文件 19](#_Toc28468665)

[2. B树功能函数 22](#_Toc28468666)

[3. 图书管理功能函数 33](#_Toc28468667)

[4. 主函数 48](#_Toc28468668)

# 一、需求分析

## 1.问题描述

图书管理基本业务活动包括：对一本书的采编入库、清除库存、借阅和归还等等。试设计一个图书管理系统，将上述业务活动借助于计算机系统完成。

## 2.基本要求

（1）每种书的登记内容至少包括书号、书名、著者、现存量和总库存量等五项。

（2）作为演示系统，不必使用文件，全部数据可以都在内存存放。但是由于上述四项基本业务活动都是通过书号(即关键字)进行的，所以要用B树(2-3树)对书号建立索引，以获得高效率。

（3）系统应实现的操作及其功能定义如下：

①采编入库：新购入一种书，经分类和确定书号之后登记到图书账目中去。如果这种书在帐中已有，则只将总库存量增加。

②清除库存：某种书已无保留价值，将它从图书账目中注销。

③借阅：如果一种书的现存量大于零，则借出一本，登记借阅者的图书证号和归还期限。

④归还：注销对借阅者的登记，改变该书的现存量。

⑤显示：以凹入表的形式显示B树。这个操作是为了调试和维护的目的而设置的。下列B树的打印格式如下所示：

60

60 30

20

30 70,72 50,52

70,72

20 50,52 68 71 100 68

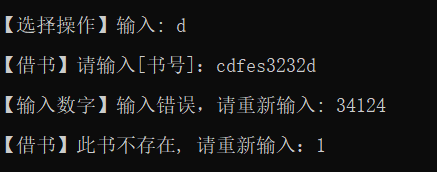
71

## 3.已完成情况

|  |  |  |  |
| --- | --- | --- | --- |
| **基本功能** | | **选做功能** | |
| **图书入库** | 已完成 | **操作日志** | 已完成 |
| **删除图书** | 已完成 | **查询作者著作** | 已完成 |
| **借书** | 已完成 | **查询书籍状态** | 已完成 |
| **还书** | 已完成 | **预约借书** | 已完成 |
| **拓展功能** | | | |
| **查询读者状态** | | 查询读者的借阅、预约列表（已完成） | |
| **总结** | | | |
| **1、图书管理系统的四个基本功能已经全部完成。**  **2、选择内容四个功能全部完成。**  **3、增加了一个拓展功能，可以更具读者借书证号查询他的借阅状态。** | | | |
|  | | | |

## 4.程序亮点

**亮点一：**系统输入排错功能完善，处理用户的乱输入乱操作，增加了系统的稳健性。

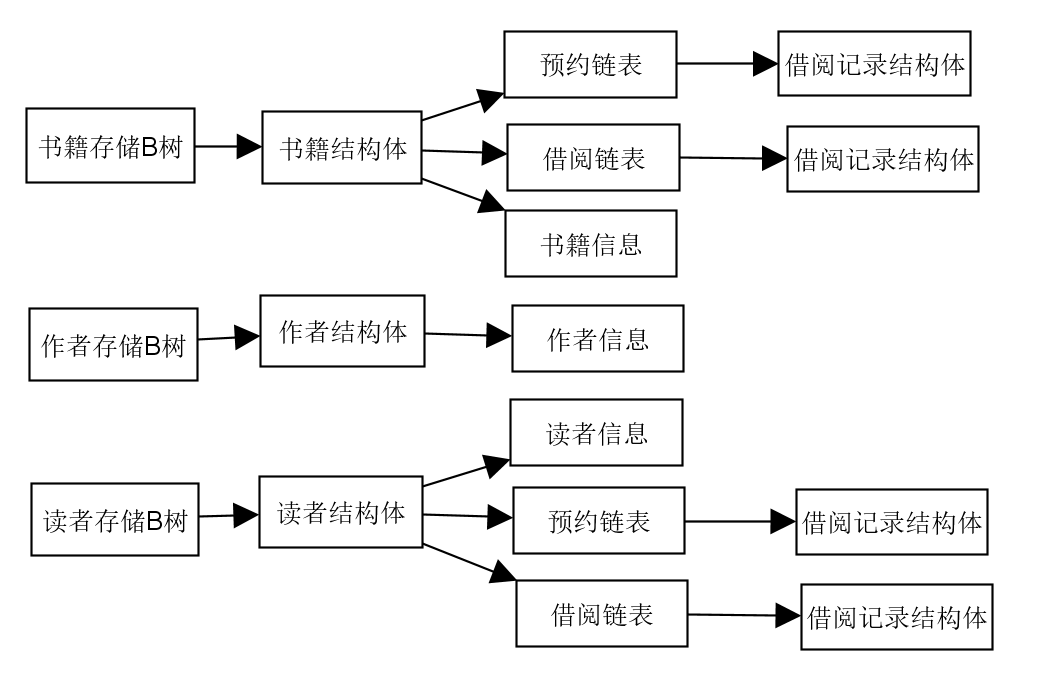


**亮点二：**系统功能有序，使用功能界面完善，用户体验效果好

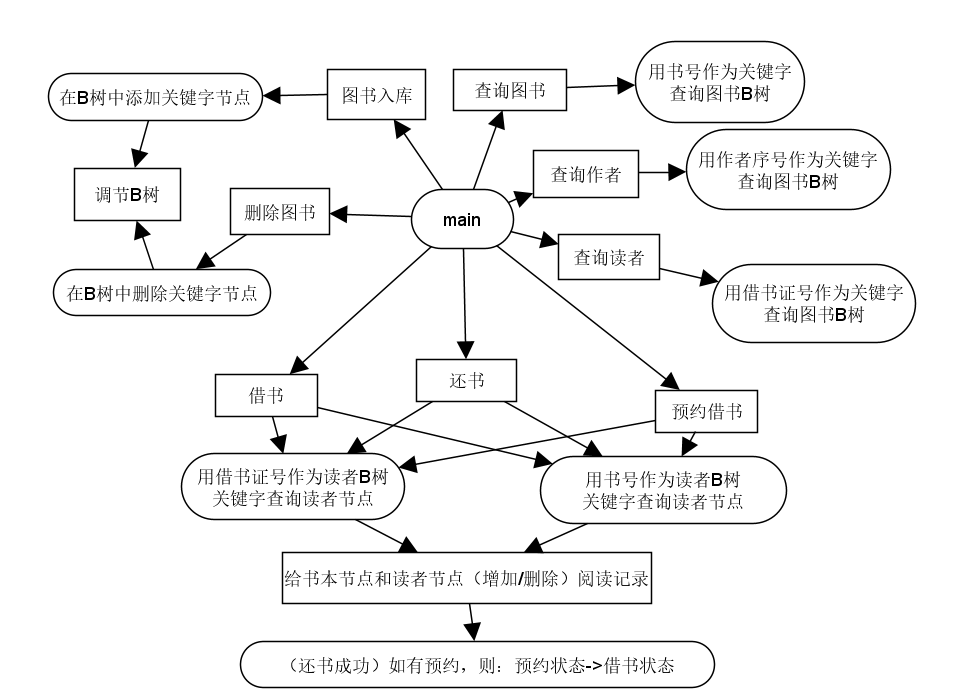


# 二、概要设计

## 1. 数据类型的定义

****

## 2.功能模块关系



# 三、 详细设计

## 1. 数据类型定义

typedef struct{ //被借阅的书籍

int tag; //状态： 0->预约，1->借阅

int readerId; //读者id

int borrowTime; //出借时间

int backTime; //归还时间

char readerName[30]; //读者名字

char bookName[30]; //书籍名字

}LendBook, \*LendBookP;

typedef struct linkNode {

int type; //类型：0->借阅记录、1->操作记录、3->作家的书

LendBook \*lenBook; //借阅记录

char \*log; //操作记录

char \*book; //作家的书

struct linkNode \*next;

}LinkNode, \*LinkType;

typedef struct{ //书籍

int id; //书籍id

char name[30]; //书名

int authorId; //作者id

char authorName[30]; //作者

int totalNum; //书籍数量

int stockNum; //书籍库存

LinkNode \*borrowList; //借阅列表

LinkNode \*appointList; //预约列表

}Book;

typedef struct{ //作者

int id; //作者id

char name[30]; //作者名字

LinkNode \*bookList; //出版书籍

}Author;

typedef struct{ //读者

int id; //读者id

char name[30]; //读者名字

LinkNode \*borrowList; //借阅列表

LinkNode \*appointList; //预约列表

}Reader;

typedef struct rcdNode{

int type; //结构体数据类型：

//0->书籍、1->作者、2->读者

Book \*book; //书籍

Author \*author; //作者

Reader \*reader; //读者

}RcdNode, \*RcdType;

typedef struct BTNode{ //B树和B树结点类型

int keynum; //结点关键字个数

KeyType key[MAXM]; //关键字数组，key[0]不使用

BTNode \*parent; //双亲结点指针

BTNode \*ptr[MAXM]; //孩子结点指针数组

RcdNode \*rcd[MAXM];

}BTNode,\*BTree;

typedef struct{ //B树查找结果类型

BTNode \*pt; //指向找到的结点

int i; //在结点中的关键字位置;

int tag; //查找成功与否标志

}Result;

typedef struct LNode{ //链表和链表结点类型

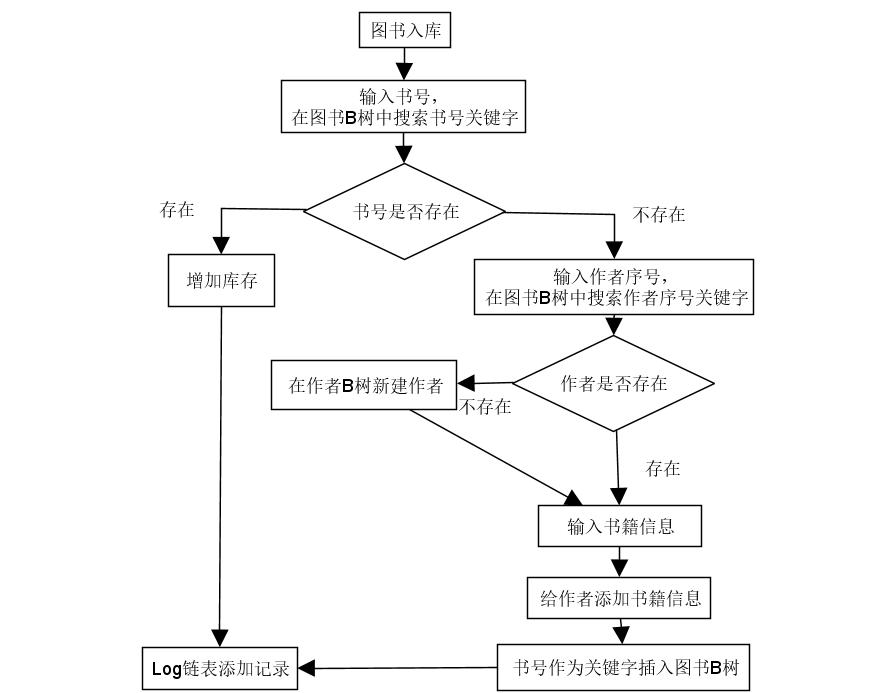
BTree data; //数据域

struct LNode \*next; //指针域

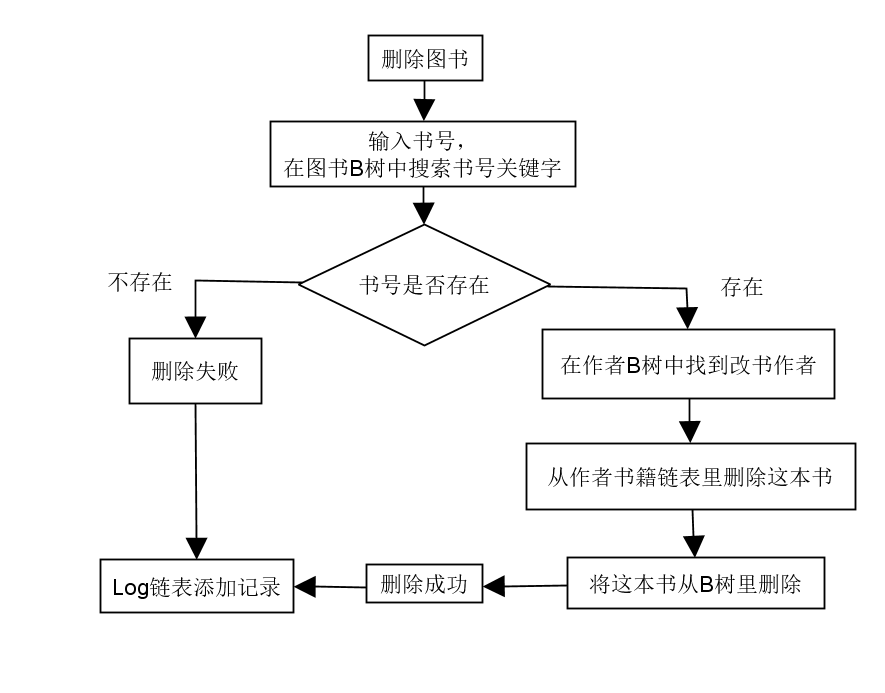
}LNode, \*LinkList;

## 2. 函数功能具体设计

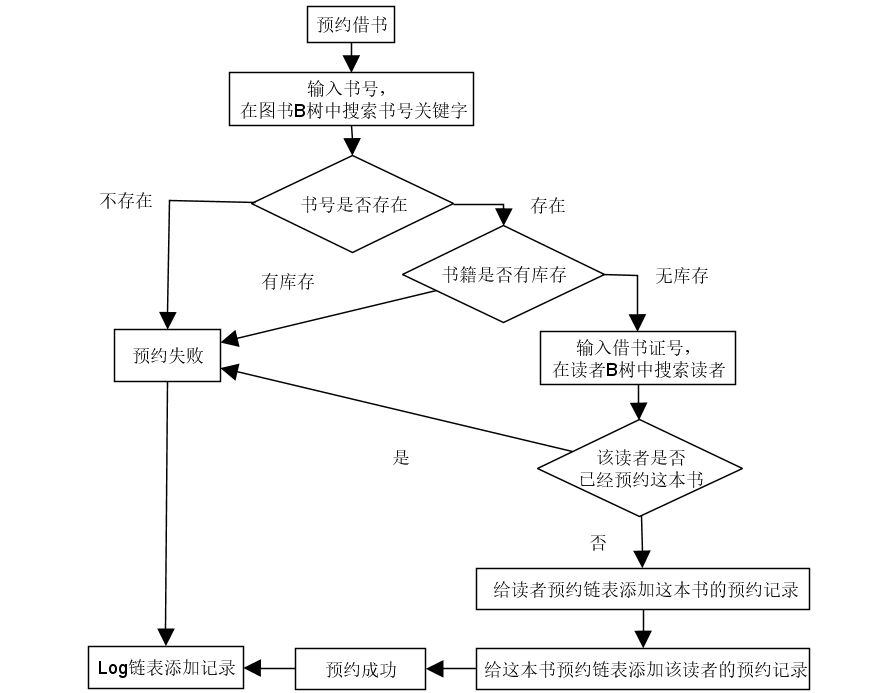
1. 图书入库



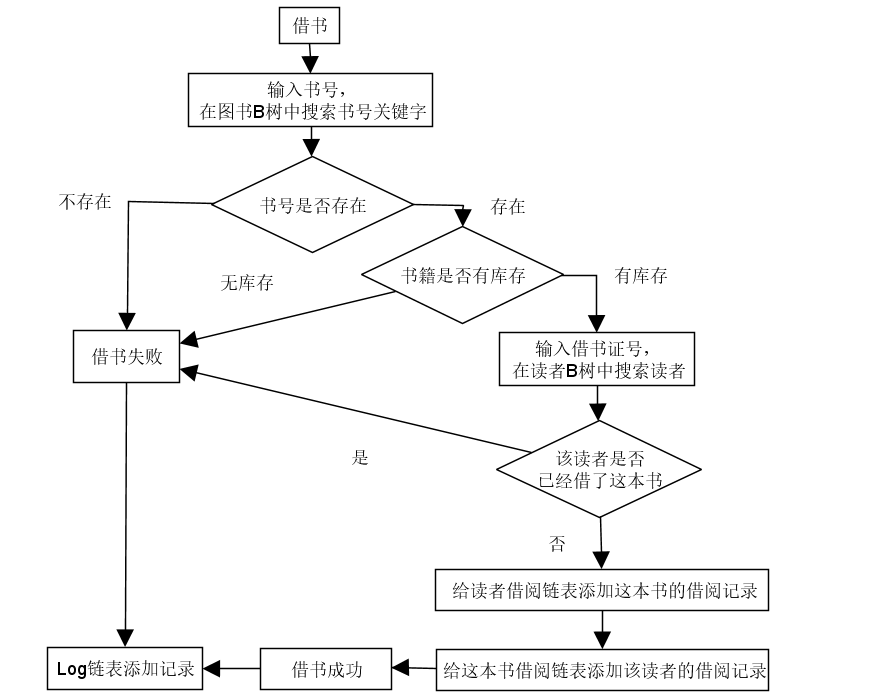
1. 删除图书



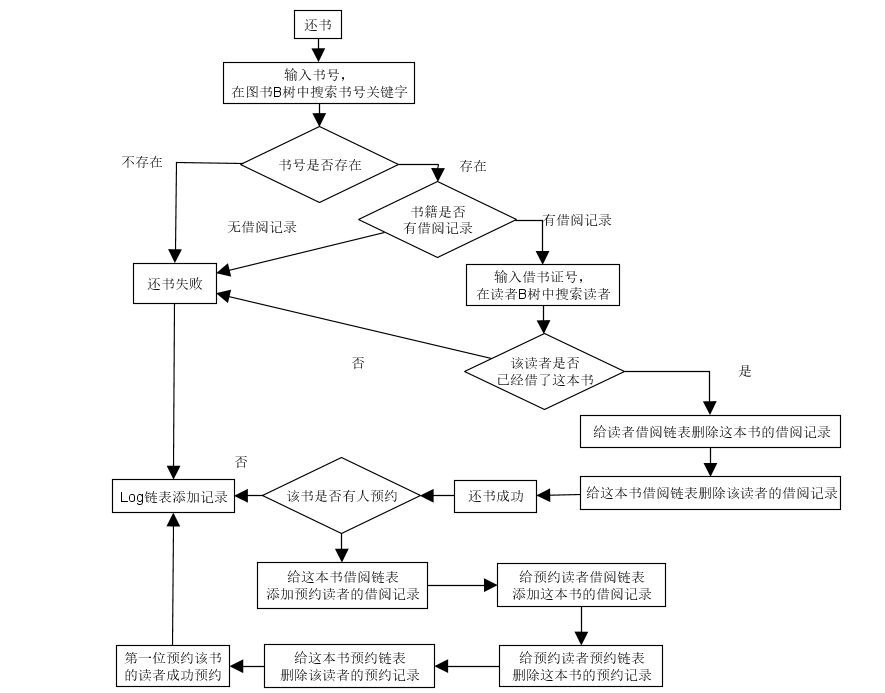
1. 预约借书



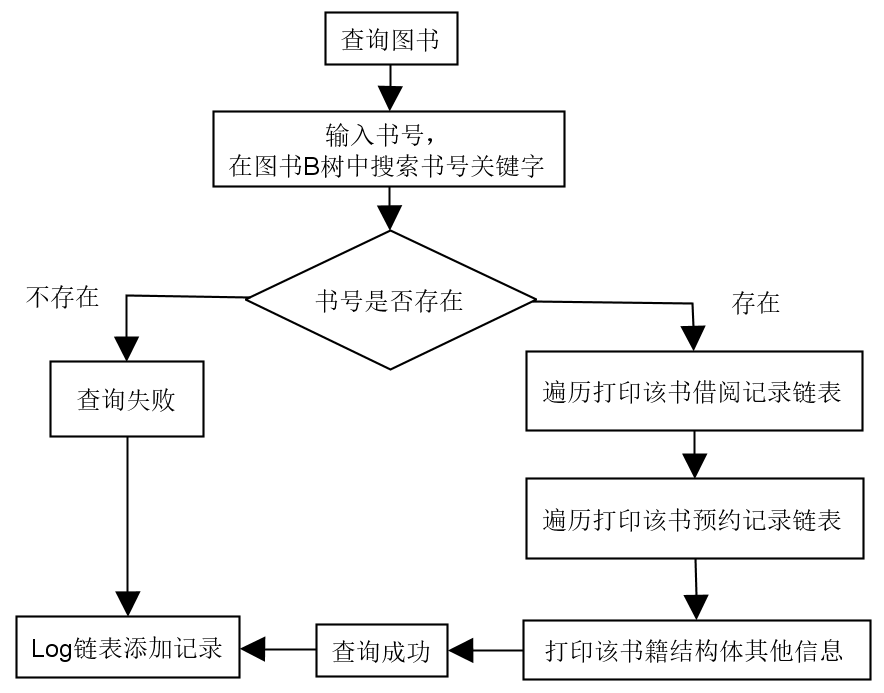
1. 借书



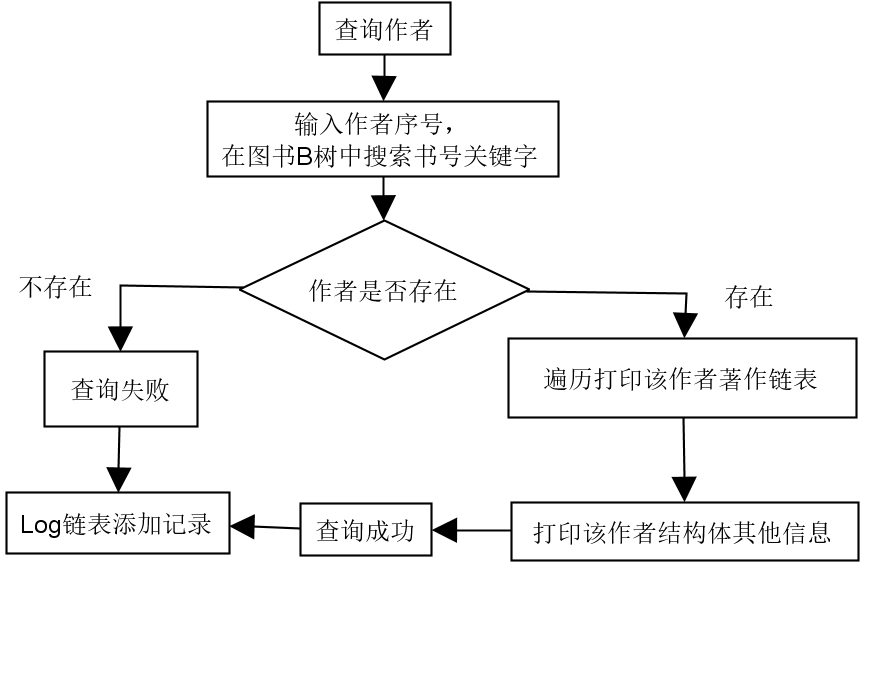
1. 还书



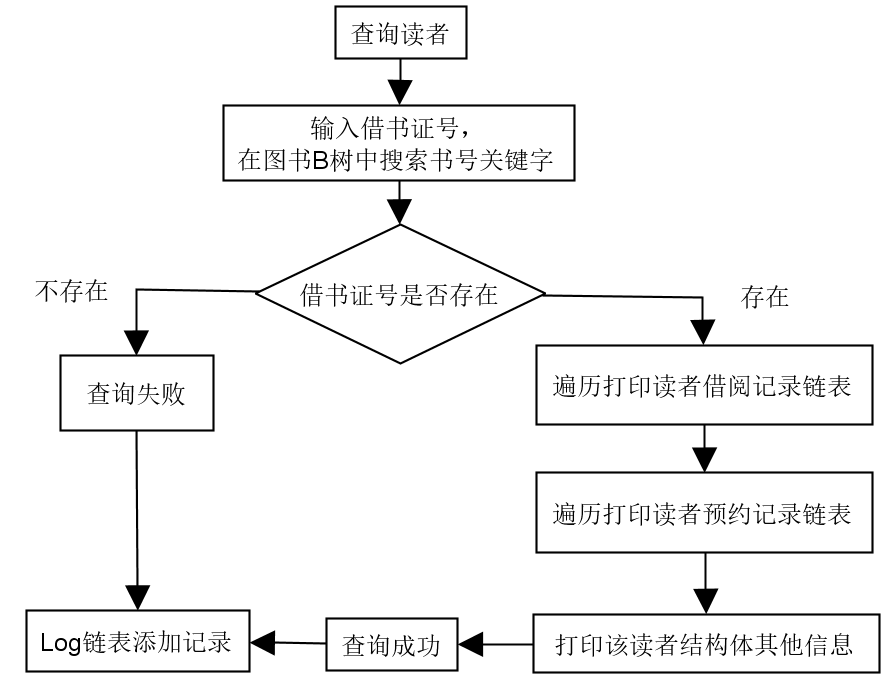
1. 查询图书



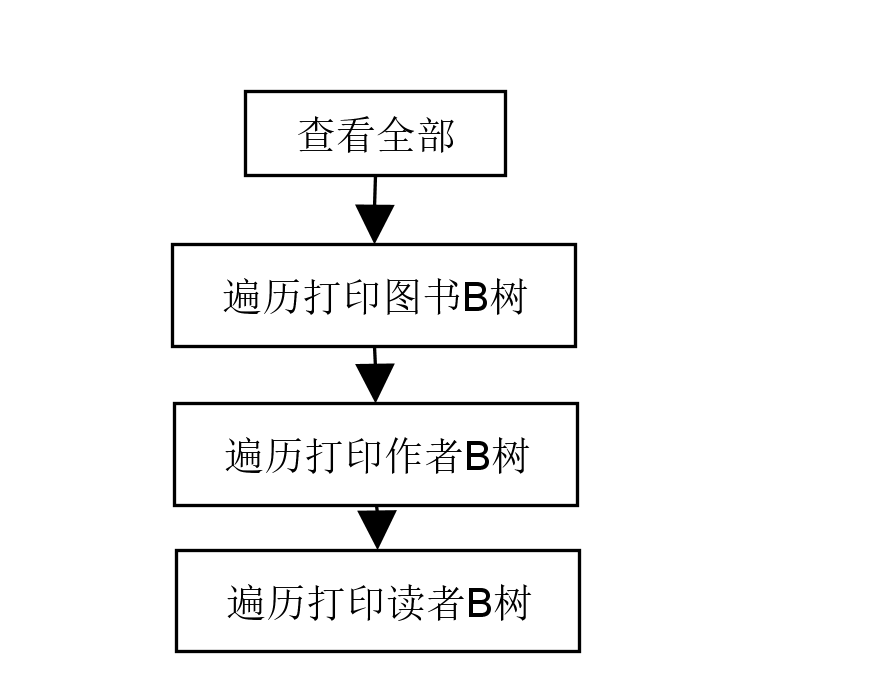
1. 查询作者



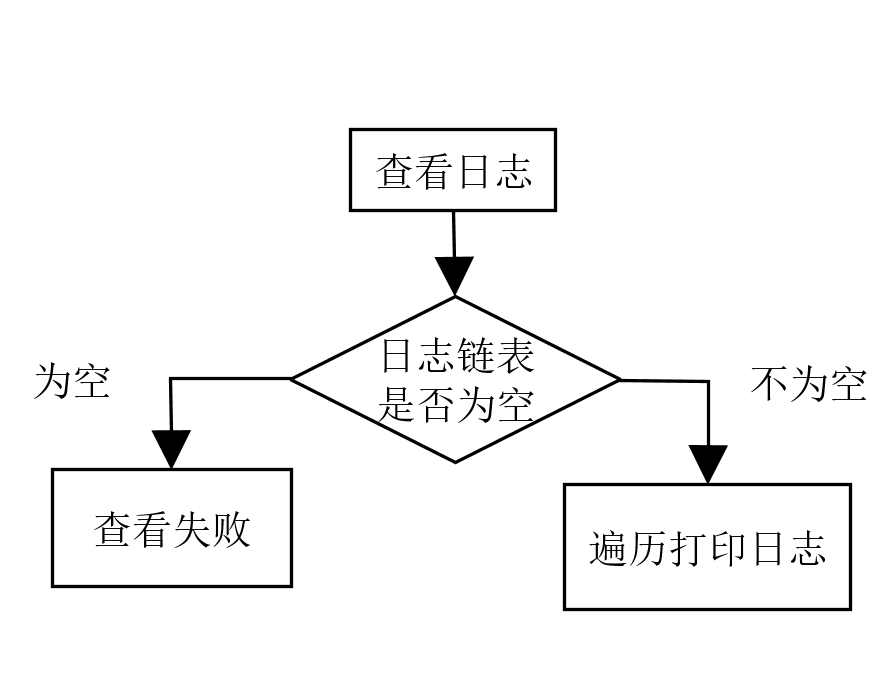
1. 查询读者



1. 查看全部（打印B树）



1. 操作日志



# 四、 调试分析

## 1. 问题分析与解决

**难题一：**图书、作者、读者数据存储定义。

一开始构思的时候，并没有打算将作者和读者也用B树存储起来，初步设想是用两个链表。但是后来想到查询作者和查询读者的查询难度和查询书籍应该一个等级，所以用B树存储的话能够提高查询速度和存储效率，所以选择了给图书、作者、读者分别用一个B树存储起来。

**难题二：**数据结构体之间的关系设计。

在图书、作者、读者关系里面，他们互有指向又相互独立，所以三者的结构体以及对应信息的指针指向就需要好好梳理。举个例子，图书里面要有借阅信息，读者里面也要有借阅信息，而借阅信息里面有图书信息和读者信息。

结构体的定义指向不能形成闭环，所以为了实现我将图书信息里面的图书名称和读者信息里面的作者名称抽取出来，放到了借阅信息里面。而关于借阅信息链表的增删改查，关键信息也就变成了作者名称和图书名称。

**难题三：**日期的定义

无论是借书、预约、还书，都和日期有关，直接让用户输入时间的话可能引发诡异的BUG，所以，我将这个系统的日期做了简化，初始默认是第一天，每一个操作增加一天，用户借书、预约的时候都是只输入一个借书天数num，程序会将这个天数处理，所以，借书的时候以今天第x天为借书日期，还书日期就是第x+num天，而预约成功的话，就以这本书被上一个借走的人还书的那一天y为借书日期，还书日期就是y+num。当然，这个方法有些取巧，但是能简化预约功能的实现和合理化整体系统的功能。

## 2. 经验体会

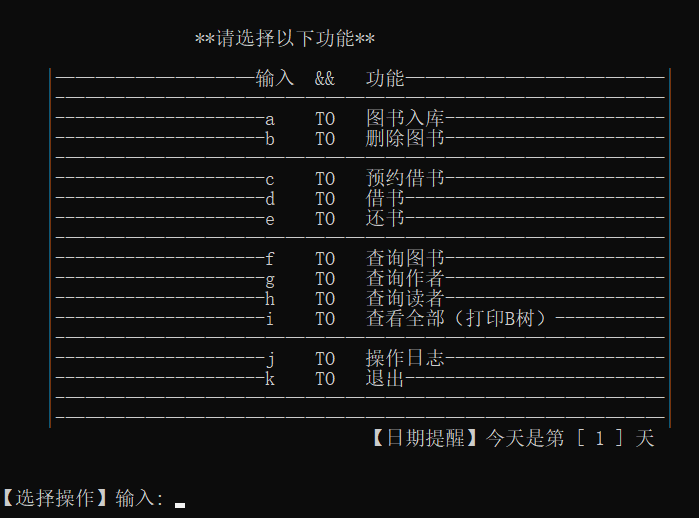
最大的体会就是，如果要设计一个整体性功能完善的系统的话，一开始一定要构思好整体的逻辑，包括各种功能的具体定义、实现的合理性、交互的实用性以及功能和功能之间整体性。功能和功能之间，关联性是很强的，一旦一个功能发生改变或者增删某一个功能的时候，其他的许多逻辑也要跟着改，这是很没必要也很浪费时间的。举个例子，我一开始并没有将读者查询、作者查询放入我的规划功能里面，所以结构体的定义都是基于图书B树来的，后来想要尝试选做内容的时候，整体上逻辑大改，结构体修改和新定义了一大堆。

再来就是，灵活debug吧，c语言的封装性比较强，很多bug都隐藏在这个指针和那个指针之间，如果不用特殊方法去检测的话是很难找出来的。所以要灵活利用断点、打印printf这些来实现程序的实时监控。

最后呢就是，坚持、耐心、细心。这三个在这次课设都充分地帮助到了我，让我在一次又一次的逻辑闭环和死debug的苦海中挣脱出来。

# 五、 用户使用说明

## 1. 功能菜单



## 2. 使用说明

**基本用法：**用户通过a~k等11个字母输入选择具体功能。

**特殊说明：**

1. 每一次使用完一个功能之后会重新清空命令行重新渲染菜单
2. 每一次使用完一个功能之后日期会增加一天
3. 日期输入采取简化输入，用户输入x，代表第x天
4. 当前日期显示在菜单右下角
5. 一个人预约一本书的话，会在上一个读者还书的时候帮他成功预约

**测试数据:**

**作者:**

序号1——"曹雪芹" 序号2——"吴承恩" 序号3——"施耐庵"

序号4——"罗贯中" 序号5——"唐家三少" 序号6——"天蚕土豆"

序号7——"江南"

**读者:**

读书证号111——"小红" 读书证号222——"小明"

读书证号333——"小华"

**书籍：（书号和作家序号相对应）**

书号1——《红楼梦》 书号2——《西游记》 书号3——《水浒传》

书号4——《三国演义》 书号5——《斗罗大陆》 书号6——《斗破苍穹》

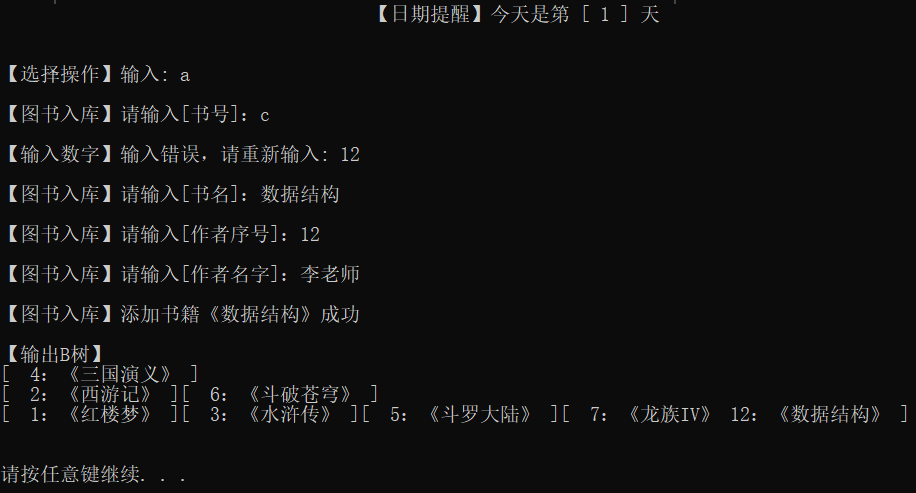
书号7——《龙族IV》

# 六、 测试结果

## 1. 图书入库

示例：加入书籍《数据结构》（书号：12）——李教授(作者序号：12)

亮点：输入非数字排错处理

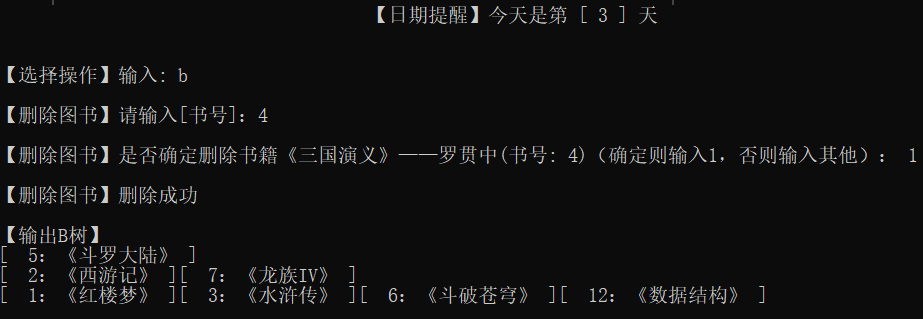


## 2. 删除图书

示例：删除书籍《三国演义》（书号：4）

亮点：输入不存在书号排错处理

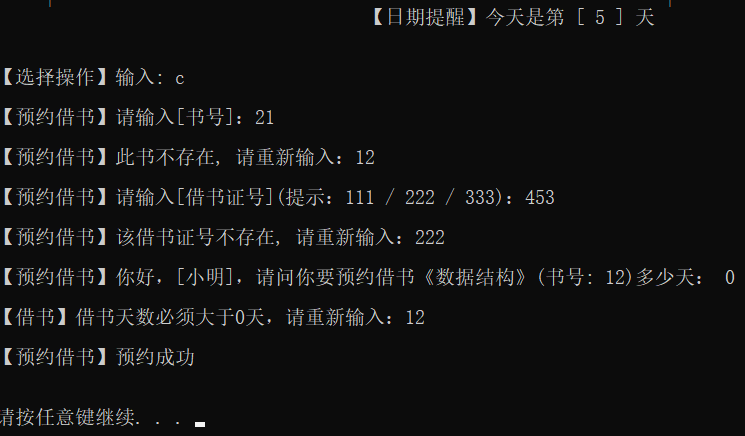




## 3. 预约借书

示例：小明（读书证号222）预约书籍《数据结构》（书号：12）

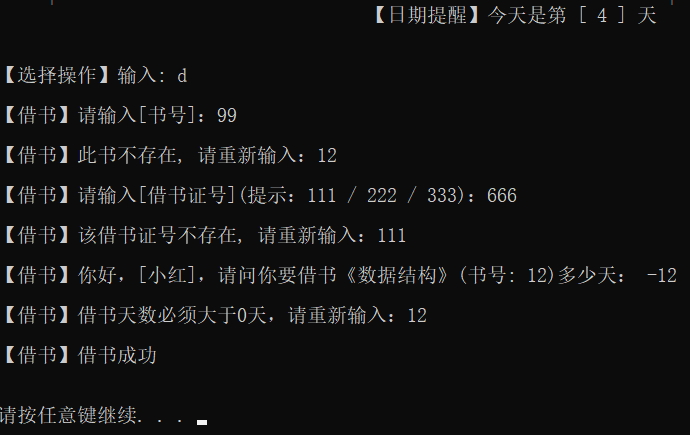
亮点：书号、借书证号、借书天数的排错处理



## 4. 借书

示例：小红（读书证号111）借阅书籍《数据结构》（书号：12）

亮点：书号、借书证号、借书天数的排错处理

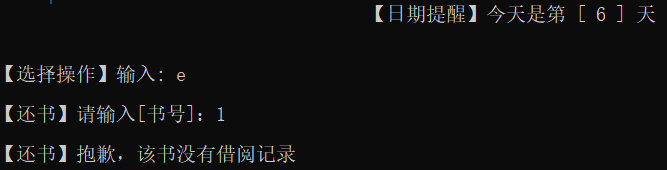


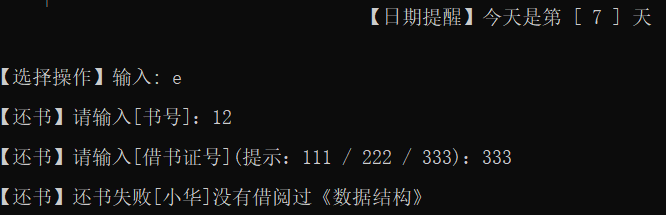
## 5. 还书

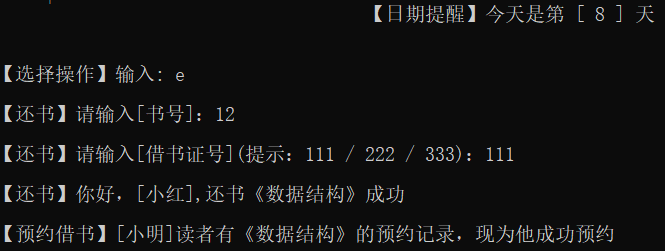
示例：小红（读书证号111）还书《数据结构》（书号：12）

亮点：

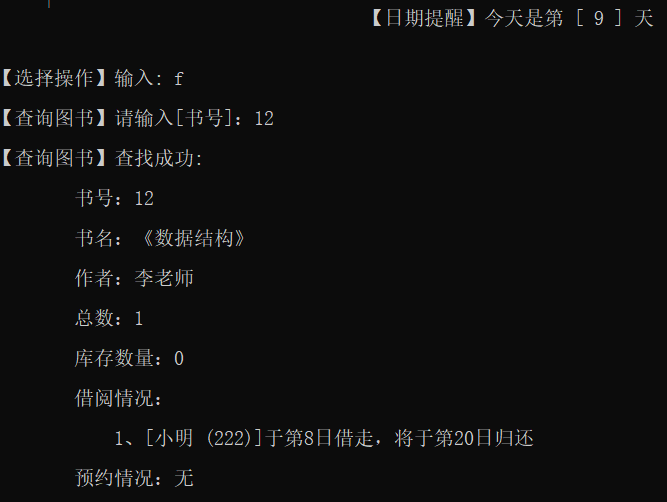
1. 没有借阅书籍读者、没有借阅记录的书籍的排错处理
2. 还书之后，该书如有预约，成功预约



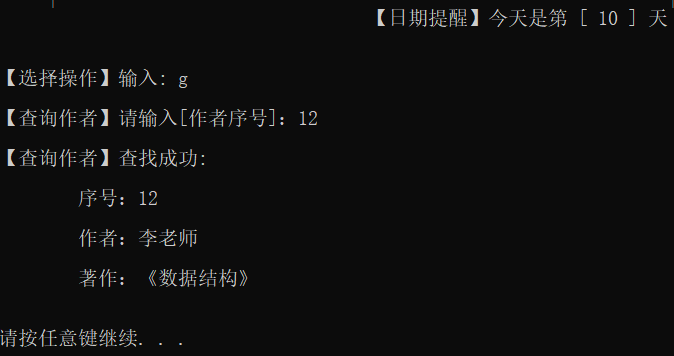




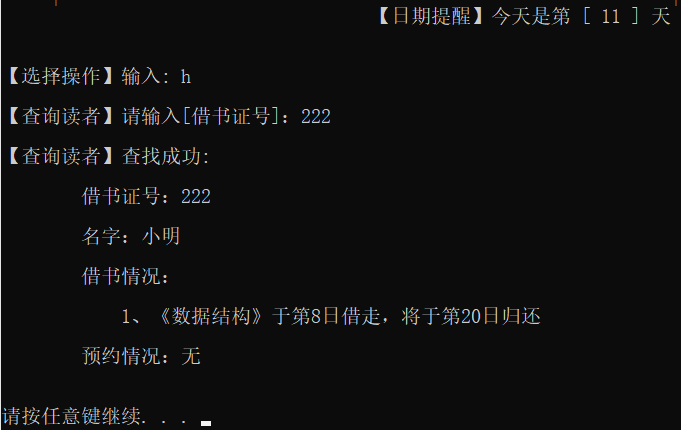
## 6. 查询图书



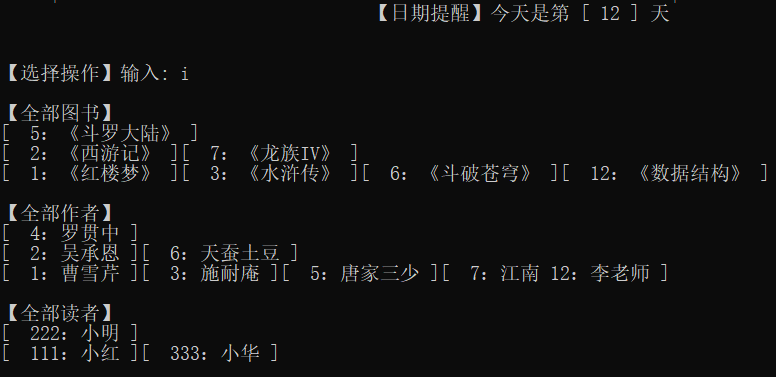
## 7. 查询作者



## 8. 查询读者



## 9. 查看全部（打印B树）



## 10. 操作日志



# 七、 附录

## 1. 头文件

#ifndef BINARYTREE\_H\_INCLUDED

#define BINARYTREE\_H\_INCLUDED

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

#define MAXM 10 //定义B树的最大的阶数

int nowDate = 1; //现在是第data天

const int m=3; //设定B树的阶数

const int Max=m-1; //结点的最大关键字数量

const int Min=(m-1)/2; //结点的最小关键字数量

typedef int KeyType; //KeyType为关键字类型

typedef struct{ //被借阅的书籍

int tag; //状态： 0->预约，1->借阅

int readerId; //读者id

int borrowTime; //出借时间

int backTime; //归还时间

char readerName[30]; //读者名字

char bookName[30]; //书籍名字

}LendBook, \*LendBookP;

typedef struct linkNode {

int type; //类型：0->借阅记录、1->操作记录、3->作家的书

LendBook \*lenBook; //借阅记录

char \*log; //操作记录

char \*book; //作家的书

struct linkNode \*next;

}LinkNode, \*LinkType;

typedef struct{ //书籍

int id; //书籍id

char name[30]; //书名

int authorId; //作者id

char authorName[30]; //作者

int totalNum; //书籍数量

int stockNum; //书籍库存

LinkNode \*borrowList; //借阅列表

LinkNode \*appointList; //预约列表

}Book;

typedef struct{ //作者

int id; //作者id

char name[30]; //作者名字

LinkNode \*bookList; //出版书籍

}Author;

typedef struct{ //读者

int id; //读者id

char name[30]; //读者名字

LinkNode \*borrowList; //借阅列表

LinkNode \*appointList; //预约列表

}Reader;

typedef struct rcdNode{

int type; //结构体数据类型：

//0->书籍、1->作者、2->读者

Book \*book; //书籍

Author \*author; //作者

Reader \*reader; //读者

}RcdNode, \*RcdType;

typedef struct BTNode{ //B树和B树结点类型

int keynum; //结点关键字个数

KeyType key[MAXM]; //关键字数组，key[0]不使用

BTNode \*parent; //双亲结点指针

BTNode \*ptr[MAXM]; //孩子结点指针数组

RcdNode \*rcd[MAXM];

}BTNode,\*BTree;

typedef struct{ //B树查找结果类型

BTNode \*pt; //指向找到的结点

int i; //在结点中的关键字位置;

int tag; //查找成功与否标志

}Result;

typedef struct LNode{ //链表和链表结点类型

BTree data; //数据域

struct LNode \*next; //指针域

}LNode, \*LinkList;

typedef enum status{ //枚举类型（依次递增）

TRUE,

FALSE,

OK,

ERROR,

OVERFLOW,

EMPTY

}Status;

#endif // BINARYTREE\_H\_INCLUDED

## 2. B树功能函数

#include "BTree.h"

Status InitBTree(BTree &t){

//初始化B树

if (t!=NULL) {

return FALSE;

}

t=NULL;

return OK;

}

int SearchBTNode(BTNode \*p,KeyType k){

//在结点p中查找关键字k的插入位置i

int i=0;

for(i=0;i<p->keynum&&p->key[i+1]<=k;i++);

return i;

}

Result SearchBTree(BTree t,KeyType k){

/\*在树t上查找关键字k,返回结果(pt,i,tag)。若查找成功,则特征值

tag=1,关键字k是指针pt所指结点中第i个关键字；否则特征值tag=0,

关键字k的插入位置为pt结点的第i个\*/

BTNode \*p=t,\*q=NULL; //初始化结点p和结点q,p指向待查结点,q指向p的双亲

int found\_tag=0; //设定查找成功与否标志

int i=0;

Result r; //设定返回的查找结果

while(p!=NULL&&found\_tag==0){

i=SearchBTNode(p,k); //在结点p中查找关键字k,使得p->key[i]<=k<p->key[i+1]

if(i>0&&p->key[i]==k) //找到待查关键字

found\_tag=1; //查找成功

else{ //查找失败

q=p;

p=p->ptr[i];

}

}

if(found\_tag==1){ //查找成功

r.pt=p;

r.i=i;

r.tag=1;

}

else{ //查找失败

r.pt=q;

r.i=i;

r.tag=0;

}

return r; //返回关键字k的位置(或插入位置)

}

void InsertBTNode(BTNode \*&p,int i,KeyType k,BTNode \*q, RcdNode \*rcd){

//将关键字k和结点q分别插入到p->key[i+1]和p->ptr[i+1]中

int j;

for(j=p->keynum;j>i;j--){ //整体后移空出一个位置

p->key[j+1]=p->key[j];

p->rcd[j+1]=p->rcd[j];

p->ptr[j+1]=p->ptr[j];

}

p->key[i+1]=k;

p->rcd[i+1]=rcd;

p->ptr[i+1]=q;

if(q!=NULL)

q->parent=p;

p->keynum++;

}

void SplitBTNode(BTNode \*&p,BTNode \*&q){

//将结点p分裂成两个结点,前一半保留,后一半移入结点q

int i;

int s=(m+1)/2;

q=(BTNode \*)malloc(sizeof(BTNode)); //给结点q分配空间

q->ptr[0]=p->ptr[s]; //后一半移入结点q

for(i=s+1;i<=m;i++){

q->key[i-s]=p->key[i];

q->rcd[i-s]=p->rcd[i];

q->ptr[i-s]=p->ptr[i];

}

q->keynum=p->keynum-s;

q->parent=p->parent;

for(i=0;i<=p->keynum-s;i++) //修改双亲指针

if(q->ptr[i]!=NULL)

q->ptr[i]->parent=q;

p->keynum=s-1; //结点p的前一半保留,修改结点p的keynum

}

void NewRoot(BTNode \*&t,KeyType k,BTNode \*p,BTNode \*q,RcdNode \*rcd){

//生成新的根结点t,原p和q为子树指针

t=(BTNode \*)malloc(sizeof(BTNode)); //分配空间

t->keynum=1;

t->ptr[0]=p;

t->ptr[1]=q;

t->key[1]=k;

t->rcd[1]=rcd;

if(p!=NULL) //调整结点p和结点q的双亲指针

p->parent=t;

if(q!=NULL)

q->parent=t;

t->parent=NULL;

}

void InsertBTree(BTree &t,int i,KeyType k,BTNode \*p, RcdNode \*rcd){

/\*在树t上结点p的key[i]与key[i+1]之间插入关键字k。若引起

结点过大,则沿双亲链进行必要的结点分裂调整,使t仍是B树\*/

BTNode \*q;

int finish\_tag,newroot\_tag,s; //设定需要新结点标志和插入完成标志

KeyType x;

RcdNode \*xRcd;

if(p==NULL) //t是空树

NewRoot(t,k,NULL,NULL,rcd); //生成仅含关键字k的根结点t

else{

x=k;

xRcd=rcd;

q=NULL;

finish\_tag=0;

newroot\_tag=0;

while(finish\_tag==0&&newroot\_tag==0){

InsertBTNode(p,i,x,q,xRcd); //将关键字x和结点q分别插入到p->key[i+1]和p->ptr[i+1]

if (p->keynum<=Max)

finish\_tag=1; //插入完成

else{

s=(m+1)/2;

SplitBTNode(p,q); //分裂结点

x=p->key[s];

xRcd=p->rcd[s];

if(p->parent){ //查找x的插入位置

p=p->parent;

i=SearchBTNode(p, x);

}

else //没找到x，需要新结点

newroot\_tag=1;

}

}

if(newroot\_tag==1) //根结点已分裂为结点p和q

NewRoot(t,x,p,q,xRcd); //生成新根结点t,p和q为子树指针

}

}

void Remove(BTNode \*p,int i){

//从p结点删除key[i]和它的孩子指针ptr[i]

int j;

for(j=i+1;j<=p->keynum;j++){ //前移删除key[i]和ptr[i]

p->key[j-1]=p->key[j];

p->rcd[j-1]=p->rcd[j];

p->ptr[j-1]=p->ptr[j];

}

p->keynum--;

}

void Substitution(BTNode \*p,int i){

//查找被删关键字p->key[i](在非叶子结点中)的替代叶子结点(右子树中值最小的关键字)

BTNode \*q;

for(q=p->ptr[i];q->ptr[0]!=NULL;q=q->ptr[0]);

p->key[i]=q->key[1]; //复制关键字值

p->rcd[i]=q->rcd[1];

}

void MoveRight(BTNode \*p,int i){

/\*将双亲结点p中的最后一个关键字移入右结点q中

将左结点aq中的最后一个关键字移入双亲结点p中\*/

int j;

BTNode \*q=p->ptr[i];

BTNode \*aq=p->ptr[i-1];

for(j=q->keynum;j>0;j--){ //将右兄弟q中所有关键字向后移动一位

q->key[j+1]=q->key[j];

q->rcd[j+1]=q->rcd[j];

q->ptr[j+1]=q->ptr[j];

}

q->ptr[1]=q->ptr[0]; //从双亲结点p移动关键字到右兄弟q中

q->key[1]=p->key[i];

q->rcd[1]=p->rcd[i];

q->keynum++;

p->key[i]=aq->key[aq->keynum]; //将左兄弟aq中最后一个关键字移动到双亲结点p中

p->rcd[i]=aq->rcd[aq->keynum];

p->ptr[i]->ptr[0]=aq->ptr[aq->keynum];

aq->keynum--;

}

void MoveLeft(BTNode \*p,int i){

/\*将双亲结点p中的第一个关键字移入左结点aq中，

将右结点q中的第一个关键字移入双亲结点p中\*/

int j;

BTNode \*aq=p->ptr[i-1];

BTNode \*q=p->ptr[i];

aq->keynum++; //把双亲结点p中的关键字移动到左兄弟aq中

aq->key[aq->keynum]=p->key[i];

aq->rcd[aq->keynum]=p->rcd[i];

aq->ptr[aq->keynum]=p->ptr[i]->ptr[0];

p->key[i]=q->key[1]; //把右兄弟q中的关键字移动到双亲节点p中

p->rcd[i]=q->rcd[1];

q->ptr[0]=q->ptr[1];

q->keynum--;

for(j=1;j<=q->keynum;j++){ //将右兄弟q中所有关键字向前移动一位

q->key[j]=q->key[j+1];

q->rcd[j]=q->rcd[j+1];

q->ptr[j]=q->ptr[j+1];

}

}

void Combine(BTNode \*p,int i){

/\*将双亲结点p、右结点q合并入左结点aq，

并调整双亲结点p中的剩余关键字的位置\*/

int j;

BTNode \*q=p->ptr[i];

BTNode \*aq=p->ptr[i-1];

aq->keynum++; //将双亲结点的关键字p->key[i]插入到左结点aq

aq->key[aq->keynum]=p->key[i];

aq->rcd[aq->keynum]=p->rcd[i];

aq->ptr[aq->keynum]=q->ptr[0];

for(j=1;j<=q->keynum;j++){ //将右结点q中的所有关键字插入到左结点aq

aq->keynum++;

aq->key[aq->keynum]=q->key[j];

aq->rcd[aq->keynum]=q->rcd[j];

aq->ptr[aq->keynum]=q->ptr[j];

}

for(j=i;j<p->keynum;j++){ //将双亲结点p中的p->key[i]后的所有关键字向前移动一位

p->key[j]=p->key[j+1];

p->rcd[j]=p->rcd[j+1];

p->ptr[j]=p->ptr[j+1];

}

p->keynum--; //修改双亲结点p的keynum值

free(q); //释放空右结点q的空间

}

void AdjustBTree(BTNode \*p,int i){

//删除结点p中的第i个关键字后,调整B树

if(i==0) //删除的是最左边关键字

if(p->ptr[1]->keynum>Min) //右结点可以借

MoveLeft(p,1);

else //右兄弟不够借

Combine(p,1);

else if(i==p->keynum) //删除的是最右边关键字

if(p->ptr[i-1]->keynum>Min) //左结点可以借

MoveRight(p,i);

else //左结点不够借

Combine(p,i);

else if(p->ptr[i-1]->keynum>Min) //删除关键字在中部且左结点够借

MoveRight(p,i);

else if(p->ptr[i+1]->keynum>Min) //删除关键字在中部且右结点够借

MoveLeft(p,i+1);

else //删除关键字在中部且左右结点都不够借

Combine(p,i);

}

int FindBTNode(BTNode \*p,KeyType k,int &i){

//反映是否在结点p中是否查找到关键字k

if(k<p->key[1]){ //结点p中查找关键字k失败

i=0;

return 0;

}

else{ //在p结点中查找

i=p->keynum;

while(k<p->key[i]&&i>1)

i--;

if(k==p->key[i]) //结点p中查找关键字k成功

return 1;

}

return 0;

}

int BTNodeDelete(BTNode \*p,KeyType k){

//在结点p中查找并删除关键字k

int i;

int found\_tag; //查找标志

if(p==NULL)

return 0;

else{

found\_tag=FindBTNode(p,k,i); //返回查找结果

if(found\_tag==1){ //查找成功

if(p->ptr[i-1]!=NULL){ //删除的是非叶子结点

Substitution(p,i); //寻找相邻关键字(右子树中最小的关键字)

BTNodeDelete(p->ptr[i],p->key[i]); //执行删除操作

}

else

Remove(p,i); //从结点p中位置i处删除关键字

}

else

found\_tag=BTNodeDelete(p->ptr[i],k); //沿孩子结点递归查找并删除关键字k

if(p->ptr[i]!=NULL)

if(p->ptr[i]->keynum<Min) //删除后关键字个数小于MIN

AdjustBTree(p,i); //调整B树

return found\_tag;

}

}

void BTreeDelete(BTree &t,KeyType k){

//构建删除框架，执行删除操作

BTNode \*p;

int a=BTNodeDelete(t,k); //删除关键字k

if(a==0) {} //查找失败

else if(t->keynum==0){ //调整

p=t;

t=t->ptr[0];

free(p);

}

}

void DestroyBTree(BTree &t){

//递归释放B树

int i;

BTNode\* p=t;

if(p!=NULL){ //B树不为空

for(i=0;i<=p->keynum;i++){ //递归释放每一个结点

DestroyBTree(\*&p->ptr[i]);

}

free(p);

}

t=NULL;

}

Status InitQueue(LinkList &L){

//初始化队列

L=(LNode\*)malloc(sizeof(LNode)); //分配结点空间

if(L==NULL) //分配失败

return OVERFLOW;

L->next=NULL;

return OK;

}

LNode\* CreateNode(BTNode \*p){

//新建一个结点

LNode \*q;

q=(LNode\*)malloc(sizeof(LNode)); //分配结点空间

if(q!=NULL){ //分配成功

q->data=p;

q->next=NULL;

}

return q;

}

Status Enqueue(LNode \*p,BTNode \*q){

//元素q入队列

if(p==NULL)

return ERROR;

while(p->next!=NULL) //调至队列最后

p=p->next;

p->next=CreateNode(q); //生成结点让q进入队列

return OK;

}

Status Dequeue(LNode \*p,BTNode \*&q){

//出队列，并以q返回值

LNode \*aq;

if(p==NULL||p->next==NULL) //删除位置不合理

return ERROR;

aq=p->next; //修改被删结点aq的指针域

p->next=aq->next;

q=aq->data;

free(aq); //释放结点aq

return OK;

}

Status IfEmpty(LinkList L){

//队列判空

if(L==NULL) //队列不存在

return ERROR;

if(L->next==NULL) //队列为空

return TRUE;

return FALSE; //队列非空

}

void DestroyQueue(LinkList L){

//销毁队列

LinkList p;

if(L!=NULL){

p=L;

L=L->next;

free(p); //逐一释放

DestroyQueue(L);

}

}

Status Traverse(BTree t,LinkList L,int newline,int sum){

//用队列遍历输出B树

int i;

BTree p;

if(t!=NULL){

printf("[ ");

Enqueue(L,t->ptr[0]); //入队

for(i=1;i<=t->keynum;i++){

printf(" %d：",t->key[i]);

if(t->rcd[i]->type == 0) {

printf("《%s》",t->rcd[i]->book->name);

} else if(t->rcd[i]->type == 1) {

printf("%s",t->rcd[i]->author->name);

} else if(t->rcd[i]->type == 2) {

printf("%s",t->rcd[i]->reader->name);

}

Enqueue(L,t->ptr[i]); //子结点入队

}

sum+=t->keynum+1;

printf(" ]");

if(newline==0){ //需要另起一行

printf("\n");

newline=sum-1;

sum=0;

}

else

newline--;

}

if(IfEmpty(L)==FALSE){ //l不为空

Dequeue(L,p); //出队，以p返回

Traverse(p,L,newline,sum); //遍历出队结点

}

return OK;

}

Status PrintBTree(BTree t){

//输出B树

LinkList L;

if(t==NULL){

printf(" B树为空树");

return OK;

}

InitQueue(L); //初始化队列

Traverse(t,L,0,0); //利用队列输出

DestroyQueue(L); //销毁队列

return OK;

}

// 初始化链表

void initLend(LinkType &list) {

list = (LinkNode \*)malloc(sizeof(LinkNode));

list->next = NULL;

}

// 添加节点,返回节点指针

LinkType addLinkNode(LinkType &list, int type) {

LinkNode \*p;

p = list;

while(p->next!=NULL) {

p = p->next;

}

p->next = (LinkNode \*)malloc(sizeof(LinkNode));

p->next->next = NULL;

p->type = type;

return p->next;

}

## 3. 图书管理功能函数

#include "BTree.h"

// 输入数字

int inputInt() {

int a, result;

fflush(stdin);

a = scanf("%d",&result);

if(a == 0)

{

printf("\n【输入数字】输入错误，请重新输入: ");

result = inputInt();

}

return result;

}

// 初始化

// 添加作者

RcdNode\* addAuthor(BTree &t, int id, char\* name) {

Result s2 = SearchBTree(t, id);

RcdNode \*rcd = (RcdNode \*)malloc(sizeof(RcdNode));

if(rcd == NULL) {

return NULL;

}

rcd->author = (Author \*)malloc(sizeof(Author));

if(rcd->author == NULL) {

return NULL;

}

rcd->type = 1;

rcd->author->id = id;

strcpy(rcd->author->name, name);

initLend(rcd->author->bookList);

InsertBTree(t, s2.i, id, s2.pt, rcd);

return rcd;

}

// 给作者添加书籍

void addAuthorBook(Author \*&author, char\* bookName) {

LinkNode \*p = addLinkNode(author->bookList, 3);

p->book = (char\*)malloc(sizeof(char)\*20);

if(p->book == NULL) {

return;

}

strcpy(p->book, bookName);

}

// 给作者删除书籍

int deleteAuthorBook(Author \*&author, char\* bookName) {

LinkNode \*p, \*pre;

p = author->bookList;

while(p->next != NULL && strcmp(p->next->book, bookName) != 0) {

p = p->next;

}

if(p->next == NULL) {

return 0; // 找不到借书记录

}

pre = p;

p = p->next;

pre->next = p->next;

free(p);

return 1;

}

// 添加书籍

int addBookFun(BTree &t, int id, char\* name, BTree &author, int authorId){

Result s = SearchBTree(t, id);

if(s.tag == 1) {

Book \*book = s.pt->rcd[s.i]->book;

book->totalNum++;

book->stockNum++;

} else {

RcdNode \*rcd = (RcdNode \*)malloc(sizeof(RcdNode));

if(rcd == NULL) {

return 0;

}

rcd->book = (Book \*)malloc(sizeof(Book));

if(rcd->book == NULL) {

return 0;

}

rcd->type = 0;

rcd->book->id = id;

strcpy(rcd->book->name, name);

//

Result s2 = SearchBTree(author, authorId);

if(s2.tag == 0) {

RcdNode \*rcd2 = addAuthor(author, authorId, "不知名作家");

addAuthorBook(rcd2->author, name);

strcpy(rcd->book->authorName, "不知名作家");

} else {

char \*authorName = s2.pt->rcd[s2.i]->author->name;

Author \*p2 = s2.pt->rcd[s2.i]->author;

addAuthorBook(p2, name);

strcpy(rcd->book->authorName, authorName);

}

//

rcd->book->totalNum = 1;

rcd->book->stockNum = 1;

rcd->book->authorId = authorId;

initLend(rcd->book->borrowList);

initLend(rcd->book->appointList);

InsertBTree(t, s.i, id, s.pt, rcd);

}

return 1;

}

// 添加读者

void addReader(BTree &t, int id, char\* name) {

Result s = SearchBTree(t, id);

if(s.tag == 1) {

Reader \*reader = s.pt->rcd[s.i]->reader;

} else {

RcdNode \*rcd = (RcdNode \*)malloc(sizeof(RcdNode));

if(rcd == NULL) {

return;

}

rcd->reader = (Reader \*)malloc(sizeof(Reader));

if(rcd->reader == NULL) {

return;

}

rcd->type = 2;

rcd->reader->id = id;

strcpy(rcd->reader->name, name);

initLend(rcd->reader->borrowList);

initLend(rcd->reader->appointList);

InsertBTree(t, s.i, id, s.pt, rcd);

}

}

void init(BTree &book, BTree &author, BTree &reader, LinkNode \*&Log) {

//初始化作者

addAuthor(author, 1, "曹雪芹");

addAuthor(author, 2, "吴承恩");

addAuthor(author, 3, "施耐庵");

addAuthor(author, 4, "罗贯中");

addAuthor(author, 5, "唐家三少");

addAuthor(author, 6, "天蚕土豆");

addAuthor(author, 7, "江南");

//初始化书籍

addBookFun(book, 1, "红楼梦", author, 1);

addBookFun(book, 1, "红楼梦", author, 1);

addBookFun(book, 1, "红楼梦", author, 1);

addBookFun(book, 1, "红楼梦", author, 1);

addBookFun(book, 1, "红楼梦", author, 1);

addBookFun(book, 2, "西游记", author, 2);

addBookFun(book, 3, "水浒传", author, 3);

addBookFun(book, 4, "三国演义", author, 4);

addBookFun(book, 5, "斗罗大陆", author, 5);

addBookFun(book, 6, "斗破苍穹", author, 6);

addBookFun(book, 7, "龙族IV——悼亡者归来", author, 7);

addBookFun(book, 8, "龙族II——悼亡者之瞳", author, 7);

addBookFun(book, 9, "龙族I——火之晨曦", author, 7);

//初始化读者

addReader(reader, 111, "小红");

addReader(reader, 222, "小明");

addReader(reader, 333, "小华");

//初始化日志

initLend(Log);

}

// 操作日志

void addLog(LinkType &list, char \*str) {

LinkNode \*p = addLinkNode(list, 1);

p->log = (char\*)malloc(sizeof(char)\*50);

if(p->log == NULL) {

return;

}

strcpy(p->log, str);

}

void log(LinkNode \*&Log) {

int i = 1;

LinkNode \*p;

p = Log->next;

if(p == NULL) {

printf("\n【操作日志】暂无\n");

return;

}

printf("\n【操作日志】:\n ");

while(p!=NULL) {

printf("\n %d、", i);

printf(" %s\n", p->log);

i++;

p = p->next;

}

printf("\n");

}

// 图书入库

void addBook(BTree &t, BTree &author, LinkNode \*&Log) {

int id, authorId;

char name[30], authorName[30], str[50];

printf("\n【图书入库】请输入[书号]：");

id = inputInt();

Result s = SearchBTree(t, id);

if(s.tag == 1) {

Book \*book = s.pt->rcd[s.i]->book;

printf("\n【图书入库】书籍《%s》——%s(书号: %d) 已经存在，是否增加库存数量（确定则输入1，否则输入其他）： ", book->name, book->authorName, book->id);

if(inputInt() == 1) {

book->totalNum++;

book->stockNum++;

printf("\n【图书入库】添加成功，书籍《%s》数量为 %d\n", book->name, book->totalNum);

sprintf(str, "第[%d]天，图书《%s》库存数量+1", nowDate, book->name);

} else {

printf("\n【图书入库】取消成功\n");

sprintf(str, "第[%d]天，取消图书《%s》库存数量增加", nowDate, book->name);

}

} else {

printf("\n【图书入库】请输入[书名]：");

fflush(stdin);

gets(name);

printf("\n【图书入库】请输入[作者序号]：");

authorId = inputInt();

Result s2 = SearchBTree(author, authorId);

if(s2.tag == 0) {

printf("\n【图书入库】请输入[作者名字]：");

fflush(stdin);

scanf("%s", authorName);

addAuthor(author, authorId, authorName);

}

if(1 == addBookFun(t, id, name, author, authorId) ) {

printf("\n【图书入库】添加书籍《%s》成功\n", name);

sprintf(str, "第[%d]天，添加书籍《%s》成功", nowDate, name);

} else {

printf("\n【图书入库】内存空间分配失败\n");

}

}

addLog(Log, str);

}

// 删除图书

void deleteBook(BTree &t, BTree &author, LinkNode \*&Log) {

int id;

printf("\n【删除图书】请输入[书号]：");

id = inputInt();

Result s = SearchBTree(t, id);

if(s.tag == 0) {

printf("\n【删除图书】此书不存在\n");

char str[50];

sprintf(str, "第[%d]天，删除图书[%d]失败，此书不存在", nowDate, id);

addLog(Log, str);

} else {

Book \*book = s.pt->rcd[s.i]->book;

printf("\n【删除图书】是否确定删除书籍《%s》——%s(书号: %d)（确定则输入1，否则输入其他）： ", book->name, book->authorName, book->id);

if(inputInt() == 1) {

if(book->borrowList != NULL) {

free(book->borrowList);

}

if(book->appointList != NULL) {

free(book->appointList);

}

Result s2 = SearchBTree(author, book->authorId);

if (0 == deleteAuthorBook(s2.pt->rcd[s2.i]->author, book->name) ) {

printf("\n\n找不到该书记录\n\n");

}

char str[50];

sprintf(str, "第[%d]天，删除图书《%s》成功", nowDate, book->name);

addLog(Log, str);

free(book);

free(s.pt->rcd[s.i]);

BTreeDelete(t, id);

printf("\n【删除图书】删除成功\n");

} else {

printf("\n【删除图书】取消成功\n");

char str[50];

sprintf(str, "第[%d]天，取消删除图书《%s》", nowDate, book->name);

addLog(Log, str);

}

}

}

// 添加借阅记录节点

void addLendBook(LinkType &list, int readerId, char \*readerName, char \*bookName, int tag, int borrowTime, int backTime) {

LinkNode \*p = addLinkNode(list, 0);

if(p == NULL) {

return;

}

p->lenBook = (LendBook\*)malloc(sizeof(LendBook));

if(p->lenBook == NULL) {

return;

}

p->lenBook->tag = tag;

p->lenBook->readerId = readerId;

p->lenBook->borrowTime = borrowTime;

p->lenBook->backTime = backTime;

strcpy(p->lenBook->readerName, readerName);

strcpy(p->lenBook->bookName, bookName);

}

// 借书

// 匹配书籍

Result searchBook(BTree t, char \*str) {

int id;

id = inputInt();

Result s = SearchBTree(t, id);

if(s.tag == 0) {

printf("\n【%s】此书不存在, 请重新输入：", str);

s = searchBook(t, str);

}

return s;

}

// 匹配读者

Result searchReader(BTree t, char \*str) {

int id;

id = inputInt();

Result s = SearchBTree(t, id);

if(s.tag == 0) {

printf("\n【%s】该借书证号不存在, 请重新输入：", str);

s = searchReader(t, str);

}

return s;

}

// 搜索借阅记录

LinkType findLinkNode(LinkType &list, char \*name, int type) {

LinkNode \*p;

p = list;

// type: 搜索类型：

// 0->在书的阅读记录搜索读者

// 1->在书的阅读记录搜索书籍

if(type == 0) {

while(p->next != NULL && strcmp(p->next->lenBook->readerName, name) != 0) {

p = p->next;

}

} else {

while(p->next != NULL && strcmp(p->next->lenBook->bookName, name) != 0) {

p = p->next;

}

}

if(p->next == NULL) {

return NULL; // 找不到，返回空

} else {

return p; // 找到，返回上一个节点指针

}

}

// 删除借阅记录节点

int deleteLendBook(LinkType &list, char \*name, int type) {

LinkNode \*p, \*pre;

p = findLinkNode(list, name, type);

if(p == NULL) {

return 0;

}

free(p->next->lenBook);

pre = p;

p = p->next;

pre->next = p->next;

free(p);

return 1;

}

// 还书之后成功预约

int appointSuccess(Book \*&book, BTree &t) {

LinkNode \*appoint = book->appointList;

if(appoint->next == NULL) {

return 0;

}

LinkNode \*p = appoint->next;

LendBook \*lend = p->lenBook;

int readerId = lend->readerId;

Result s = SearchBTree(t, readerId);

Reader \*reader = s.pt->rcd[s.i]->reader;

printf("\n【预约借书】[%s]读者有《%s》的预约记录，现为他成功预约\n", lend->readerName, lend->bookName);

addLendBook(book->borrowList, readerId, lend->readerName, lend->bookName, 1, nowDate, nowDate+lend->backTime);

deleteLendBook(reader->appointList, book->name, 1);

addLendBook(reader->borrowList, readerId, lend->readerName, lend->bookName, 1, nowDate, nowDate+lend->backTime);

free(p->lenBook);

appoint->next = p->next;

free(p);

return 1;

}

// 天数输入

int dataNumInput() {

int res = inputInt();

if(res <= 0) {

printf("\n【借书】借书天数必须大于0天，请重新输入：");

res = dataNumInput();

}

return res;

}

// 预约

void appointBook(BTree &t, BTree &reader, LinkNode \*&Log) {

printf("\n【预约借书】请输入[书号]：");

Result s = searchBook(t, "预约借书");

Book \*book = s.pt->rcd[s.i]->book;

if(book->stockNum > 0 ) {

printf("\n【预约借书】无需预约，该书仍有库存，您可直接借书\n");

char str[50];

sprintf(str, "第[%d]天，预约借书《%s》失败, 该书仍有库存", nowDate, book->name);

addLog(Log, str);

} else {

printf("\n【预约借书】请输入[借书证号](提示：111 / 222 / 333)：");

Result s2 = searchReader(reader, "预约借书");

Reader \*p = s2.pt->rcd[s2.i]->reader;

if(findLinkNode(p->appointList, book->name, 1) != NULL) {

printf("\n【预约借书】抱歉[%s]，不能重复预约借书《%s》\n", p->name, book->name);

char str[50];

sprintf(str, "第[%d]天，[%s]预约借书《%s》失败，重复预约借书", nowDate, p->name, book->name);

addLog(Log, str);

return;

}

printf("\n【预约借书】你好，[%s]，请问你要预约借书《%s》(书号: %d)多少天： ", p->name, book->name, book->id);

int dataNum = dataNumInput();

addLendBook(book->appointList, p->id, p->name, book->name, 0, 0, dataNum);

addLendBook(p->appointList, p->id, p->name, book->name, 0, 0, dataNum);

printf("\n【预约借书】预约成功\n");

char str[50];

sprintf(str, "第[%d]天，[%s]预约借书《%s》成功", nowDate, p->name, book->name);

addLog(Log, str);

}

}

// 借书

void borrowBook(BTree &t, BTree &reader, LinkNode \*&Log) {

printf("\n【借书】请输入[书号]：");

Result s = searchBook(t, "借书");

Book \*book = s.pt->rcd[s.i]->book;

if(book->stockNum <= 0 ) {

printf("\n【借书】抱歉该书已经没有库存，若要借书烦请预约\n");

char str[50];

sprintf(str, "第[%d]天，借书《%s》失败, 没有库存", nowDate, book->name);

addLog(Log, str);

} else {

printf("\n【借书】请输入[借书证号](提示：111 / 222 / 333)：");

Result s2 = searchReader(reader, "借书");

Reader \*p = s2.pt->rcd[s2.i]->reader;

if(findLinkNode(p->borrowList, book->name, 1) != NULL) {

printf("\n【借书】抱歉[%s]，不能重复借书《%s》\n", p->name, book->name);

char str[50];

sprintf(str, "第[%d]天，[%s]借书《%s》失败，重复借书", nowDate, p->name, book->name);

addLog(Log, str);

return;

}

printf("\n【借书】你好，[%s]，请问你要借书《%s》(书号: %d)多少天： ", p->name, book->name, book->id);

int dataNum = dataNumInput();

addLendBook(book->borrowList, p->id, p->name, book->name, 1, nowDate, nowDate+dataNum);

addLendBook(p->borrowList, p->id, p->name,book->name, 1, nowDate, nowDate+dataNum);

book->stockNum--;

printf("\n【借书】借书成功\n");

char str[50];

sprintf(str, "第[%d]天，[%s]借书《%s》成功", nowDate, p->name, book->name);

addLog(Log, str);

}

}

// 还书

void backBook(BTree &t, BTree &reader, LinkNode \*&Log) {

printf("\n【还书】请输入[书号]：");

Result s = searchBook(t, "还书");

Book \*book = s.pt->rcd[s.i]->book;

if(book->stockNum == book->totalNum ) {

printf("\n【还书】抱歉，该书没有借阅记录\n");

char str[50];

sprintf(str, "第[%d]天，还书《%s》失败，该书没有借阅记录", nowDate, book->name);

addLog(Log, str);

} else {

printf("\n【还书】请输入[借书证号](提示：111 / 222 / 333)：");

Result s2 = searchReader(reader, "借书");

Reader \*p = s2.pt->rcd[s2.i]->reader;

if (deleteLendBook(p->borrowList, book->name, 1) == 1) {

deleteLendBook(book->borrowList, p->name, 0);

book->stockNum++;

printf("\n【还书】你好，[%s],还书《%s》成功\n", p->name, book->name);

if(1 == appointSuccess(book, reader)) {

book->stockNum--;

}

char str[50];

sprintf(str, "第[%d]天，[%s]还书《%s》成功", nowDate, p->name, book->name);

addLog(Log, str);

} else {

printf("\n【还书】还书失败[%s]没有借阅过《%s》\n", p->name, book->name);

char str[50];

sprintf(str, "第[%d]天，还书失败，[%s]没有借阅过《%s》", nowDate, p->name, book->name);

addLog(Log, str);

}

}

}

// 查询图书

// 打印借阅记录

void printLendBook(LinkType &list ,int type) {

int i = 1;

LinkNode \*p;

p = list->next;

if(p == NULL) {

printf("无\n");

return;

}

printf("\n");

while(p!=NULL) {

LendBook \*lend = p->lenBook;

printf("\n %d、", i);

if(lend->backTime < nowDate && lend->tag == 1) {

printf("【已逾期】");

}

if(type == 0) {

if(lend->tag == 1) {

printf("[%s (%d)]于第%d日借走，将于第%d日归还\n", lend->readerName, lend->readerId, lend->borrowTime, lend->backTime);

} else if(p->lenBook->tag == 0) {

printf("[%s (%d)]预约借书%d天\n", lend->readerName, lend->readerId, lend->backTime - lend->borrowTime);

}

} else {

if(lend->tag == 1) {

printf("《%s》于第%d日借走，将于第%d日归还\n", lend->bookName, lend->borrowTime, lend->backTime);

} else if(p->lenBook->tag == 0) {

printf("《%s》预约借书%d天\n", lend->bookName, lend->backTime - lend->borrowTime);

}

}

i++;

p = p->next;

}

}

// 功能函数

void bookData(BTree &t, LinkNode \*&Log) {

int id;

printf("\n【查询图书】请输入[书号]：");

id = inputInt();

Result s = SearchBTree(t, id);

if(s.tag == 0) {

printf("\n【查询图书】此书不存在\n");

char str[50];

sprintf(str, "第[%d]天，查询图书[%d]失败，此书不存在", nowDate, id);

addLog(Log, str);

} else {

Book \*book = s.pt->rcd[s.i]->book;

printf("\n【查询图书】查找成功: \n");

printf("\n 书号：%d\n", book->id);

printf("\n 书名：《%s》\n", book->name);

printf("\n 作者：%s\n", book->authorName);

printf("\n 总数：%d\n", book->totalNum);

printf("\n 库存数量：%d\n", book->stockNum);

printf("\n 借阅情况：");

printLendBook(book->borrowList, 0);

printf("\n 预约情况：");

printLendBook(book->appointList, 0);

char str[50];

sprintf(str, "第[%d]天，查询图书[%d]成功——《%s》", nowDate, id, book->name);

addLog(Log, str);

}

}

// 查询作者

// 打印作者全部书籍

void printAuthorBook(LinkType &list) {

int i = 1;

LinkNode \*p;

p = list->next;

if(p == NULL) {

printf("无\n");

return;

}

while(p!=NULL) {

printf("《%s》 ", p->book);

i++;

p = p->next;

}

printf("\n");

}

// 功能函数

void authorData(BTree &t, LinkNode \*&Log) {

int id;

printf("\n【查询作者】请输入[作者序号]：");

id = inputInt();

Result s = SearchBTree(t, id);

if(s.tag == 0) {

printf("\n【查询作者】此作者不存在\n");

char str[50];

sprintf(str, "第[%d]天，查询作者[%d]失败, 该作者不存在", nowDate, id);

addLog(Log, str);

} else {

Author \*author = s.pt->rcd[s.i]->author;

printf("\n【查询作者】查找成功: \n");

printf("\n 序号：%d\n", author->id);

printf("\n 作者：%s\n", author->name);

printf("\n 著作：");

printAuthorBook(author->bookList);

char str[50];

sprintf(str, "第[%d]天，查询作者[%d]成功——%s", nowDate, id, author->name);

addLog(Log, str);

}

}

// 查询读者

void readerData(BTree &t, LinkNode \*&Log) {

int id;

printf("\n【查询读者】请输入[借书证号]：");

id = inputInt();

Result s = SearchBTree(t, id);

if(s.tag == 0) {

printf("\n【查询读者】该读者不存在\n");

char str[50];

sprintf(str, "第[%d]天，查询读者[%d]失败, 该读者不存在", nowDate, id);

addLog(Log, str);

} else {

Reader \*reader = s.pt->rcd[s.i]->reader;

printf("\n【查询读者】查找成功: \n");

printf("\n 借书证号：%d\n", reader->id);

printf("\n 名字：%s\n", reader->name);

printf("\n 借书情况：");

printLendBook(reader->borrowList, 1);

printf("\n 预约情况：");

printLendBook(reader->appointList, 1);

char str[50];

sprintf(str, "第[%d]天，查询读者[%d]成功——%s", nowDate, id, reader->name);

addLog(Log, str);

}

}

// 查看全部图书（打印B树）

void printBook(BTree &book, BTree &author, BTree &reader, LinkNode \*&Log) {

printf("\n【全部图书】\n");

PrintBTree(book);

printf("\n【全部作者】\n");

PrintBTree(author);

printf("\n【全部读者】\n");

PrintBTree(reader);

}

// 功能表

void menu()

{

printf("\n\n \*\*请选择以下功能\*\*\n\n");

printf(" |——————————输入 && 功能—————|\n");

printf(" |-———————————————————————|\n");

printf(" |---------------------a TO 图书入库----------------------|\n");

printf(" |---------------------b TO 删除图书----------------------|\n");

printf(" |-———————————————————————|\n");

printf(" |---------------------c TO 预约借书---------------------|\n");

printf(" |---------------------d TO 借书--------------------------|\n");

printf(" |---------------------e TO 还书--------------------------|\n");

printf(" |-———————————————————————|\n");

printf(" |---------------------f TO 查询图书----------------------|\n");

printf(" |---------------------g TO 查询作者----------------------|\n");

printf(" |---------------------h TO 查询读者----------------------|\n");

printf(" |---------------------i TO 查看全部（打印B树）-----|\n");

printf(" |-———————————————————————|\n");

printf(" |---------------------j TO 操作日志----------------------|\n");

printf(" |---------------------k TO 退出--------------------------|\n");

printf(" |-———————————————————————|\n");

printf(" |-———————————————————————|\n");

printf(" 【日期提醒】今天是第 [ %d ] 天\n\n", nowDate);

}

// 简化交互

void reactive()

{

printf("\n\n");

system("PAUSE");

system("cls");

nowDate++;

menu();

printf("\n【选择操作】输入: ");

}

## 4. 主函数

#include "BTree.cpp"

#include "fun.cpp"

#include "BTree.h"

/\*

int main(){

Test2();

return 0;

}

\*/

void main()

{

char i;

int k, test;

BTree book = NULL;

BTree author = NULL;

BTree reader = NULL;

LinkNode \*Log = NULL;

Result s;

init(book, author, reader, Log);

//

menu();

printf("\n【选择操作】输入: ");

while((i = getchar()) != 'k')

{

if(i == '\n')

continue;

if('a' <= i && i <= 'j')

{

switch(i)

{

case'a':

// 图书入库

addBook(book, author, Log);

printf("\n【输出B树】\n");

PrintBTree(book);

reactive();

break;

case'b':

// 删除图书

deleteBook(book, author, Log);

printf("\n【输出B树】\n");

PrintBTree(book);

reactive();

break;

case'c':

// 预约借书

appointBook(book, reader, Log);

reactive();

break;

case'd':

// 借书

borrowBook(book, reader, Log);

reactive();

break;

case'e':

// 还书

backBook(book, reader, Log);

reactive();

break;

case'f':

// 查询图书

bookData(book, Log);

reactive();

break;

case'g':

// 查看作者

authorData(author, Log);

reactive();

break;

case'h':

// 查询读者

readerData(reader, Log);

reactive();

break;

case'i':

// 查看全部（打印B树）

printBook(book, author, reader, Log);

reactive();

break;

case'j':

// 操作日志

log(Log);

reactive();

break;

}

}

else

printf("\n【选择操作】选择输入有误,请重新选择: ");

while(getchar() != '\n')

continue;

}

printf("\n\n【结束】程序已经结束，感谢您的使用！\n\n");

getchar();

}