实习报告：综合实验

线性表、平衡二叉树——简易图书管理模拟系统

班级：17计算机一班 姓名：陈新朋 学号：1725111006 完成日期：2019.1.2

1. 需求分析
2. 以平衡二叉树来存储图书馆库存的书的信息（关键字为书号），内容包括书号、书名、著者、现存量和总库存量。登记借阅信息采用下图所示的“顺序表—链表”来存储，其中顺序表存储借阅者信息，链表存储借阅者所借的各种书籍信息（至少包含书号），借阅登记信息的存储结构如下示意：

借阅者1

所借书1

所借书2

…

所借书x

借阅者2

所借书1

所借书2

…

所借书y

…

所借书1

所借书2

…

所借书z

借阅者n

所借书1

所借书2

…

所借书m

1. 用户可以随时借书或者还书或者添加新书
2. 输入菜单命令N：图书入库 B:借书 R:还书 Q：退出 S：凹入表打印现有书号:

输入字符选择功能：

1. 图书入库：先输入想要添加的书的种类数n，种类数n必须大于0；之后输入每本书书号、书名、著者、现存量和总库存量。
2. 借书：输入借书证号、想要借的书的书号。
3. 还书：输入借书证号、想要借的书的书号。
4. 凹入表打印现有书号：输入S，界面会打印出现有的书号。
5. 退出：输入Q，程序退出。
6. 用户可以以文件的形式读入库存图书的信息，在”init\_input.txt”文件中每一行的数据为一本书的数据，分别为书号、书名、著者、现存量和总库存量。最后一行是字符“#”，用于判断是否读取完成。同时，用户可以从文件中读取借阅者的相关信息，进行判断。
7. 测试数据：入库书号：ISBN 7-302-02368-9，ISBN 978-7-115-16985-3/TP，ISBN 978-7-302-03314-1，ISBN7-115-10563-4/TP·3043，ISBN 978-7-121-07479-0，ISBN 978-7-115-18809-0/TP，ISBN 978-7-04-024246-1，ISBN 7-111-12886-9，ISBN 978-7-115-19601-9/TP，ISBN 7-900183-01-9。

借书证号为081716的借阅者，先借阅10种图书各一本，后归还图书ISBN 7-302-02368-9和ISBN 978-7-121-07479-0。

借书证号为081710的借阅者，先借阅图书ISBN 978-7-121-07479-0和ISBN 978-7-302-03314-110各一本，后归还图书ISBN 978-7-121-07479-0。

1. 程序执行的命令为

1）从文件中读取数据 2）判断用户输入，执行相应图书入库、借书还书的功能

二、概要设计

1、设定树节点的抽象数据类型定义：

ADT Node

{

int height;

Node \* lchild;

Node \* rchild;

string booknum;

string bookname;

string author;

unsigned int stillhave;

unsigned int total;

};

2、设定链表节点的抽象数据类型定义：

ADT bookmsg

{

string booknum;

int borrowdate;

int returndate;

bool is;

bookmsg \*next;

};

3、设定线性表的抽象数据类型定义：

ADT borrower

{

string IDnum;

bookmsg \*borrowlist;

};

4、设定二叉平衡树的抽象数据类型定义：

ADT AVLTree

{

数据对象D={ai |ai∈Node,i=1,2,…,n,n>=0}

数据关系：R1={<ai-1 ,ai >|ai-1 ,ai ∈D,i=2,…,n}

基本操作

borrowbook(string IDnum, string booknum, int borrowdate, int returndate);

初始条件：二叉平衡树已存在

操作结果：改变对应节点的信息，将图书现有量减一。

void returnbook(string IDnum, string booknum);

初始条件：二叉平衡树已存在

操作结果：改变对应节点的信息，将图书现有量加一。

Node \*InsAVLTree(Node \*T, Node key);

操作结果：插入函数，被Insert函数内部调用。

void Insert(Node key);

操作结果：插入函数，插入一个新的节点，实现图书入库，若图书已存在则库存量增加，节点不变。

void Insertber(string name);

初始条件：二叉平衡树已存在

操作结果：将借阅者的信息加入到顺序表中进行存储。

Node \*GetPos\_true(Node \*T, string key);

操作结果：搜索平衡二叉树，为GetPos函数内部调用。

Node \*GetPos(string key);

初始条件：二叉平衡树已存在

操作结果：搜索平衡二叉树，返回关键字与key相同的节点，若找不到，则返回空。

void print();

初始条件：二叉平衡树已存在

操作结果：使用中序遍历打印二叉树。

AVLTree()

操作结果：构造函数，对平衡二叉树进行初始化。

int GetHeight(Node \*T);

初始条件：节点T已存在

操作结果：返回节点T在树中的高度。

int GetBalance(Node \* T);

初始条件：节点T已存在

操作结果：返回节点T的平衡因子。

Node \*LL\_rotate(Node \* T);

初始条件：节点T已存在

操作结果：对节点T进行左左旋转。

Node \*RR\_rotate(Node \* T);

初始条件：节点T已存在

操作结果：对节点T进行右右旋转。

Node \*LR\_rotate(Node \* T);

初始条件：节点T已存在

操作结果：对节点T进行左右旋转。

Node \*RL\_rotate(Node \* T);

初始条件：节点T已存在

操作结果：对节点T进行右左旋转。

void InOrder(Node \* node);

初始条件：二叉平衡树已存在

操作结果：中序遍历二叉平衡树，并打印数据。

};

本程序包含四个模块

1. 主程序模块

int main()

{

初始化；

While（1）

{

接受命令；

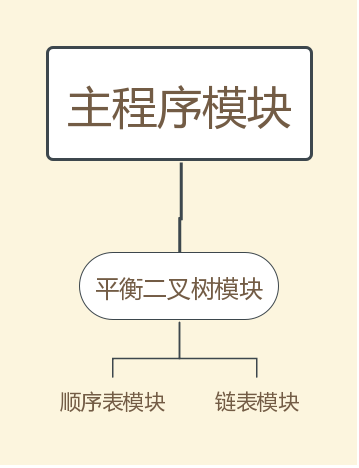
处理命令；

}

return 0;

}

1. 平衡二叉树模块——实现图书数据的存储
2. 顺序表模块——实现借阅者信息的存储
3. 链表模块——实现借阅的书籍的存储



三、详细设计

1、AVLTree.h头文件

#ifndef \_AVLTree\_H

#define \_AVLTree\_H

#include "string"

#include <iostream>

using namespace std;

struct Node

{

int height;

Node \* lchild;

Node \* rchild;

string booknum;

string bookname;

string author;

unsigned int stillhave;

unsigned int total;

Node(unsigned int stillhave = 0, unsigned int total = 0, int key = 0)

{

this->booknum = booknum;

this->bookname = bookname;

this->author = author;

this->stillhave = stillhave;

this->total = total;

this->height = 1;

this->lchild = this->rchild = NULL;

}

};

struct bookmsg

{

string booknum;

int borrowdate;

int returndate;

bool is;

bookmsg \*next;

};

struct borrower

{

string IDnum;

bookmsg \*borrowlist;

};

class AVLTree

{

public:

borrower \* borrowerlist;

void borrowbook(string IDnum, string booknum, int borrowdate, int returndate);

void returnbook(string IDnum, string booknum);

void Insert(Node key);

void Insertber(string name);

Node \*GetPos(string key);

void print();

AVLTree()

{

header = new Node;

bor = 0;

borrowerlist = new borrower[50];

}

Node \* header;

private:

int bor;//借书人总数

Node \*InsAVLTree(Node \*T, Node key);

int GetHeight(Node \*T);

int GetBalance(Node \* T);

Node \*GetPos\_true(Node \*T, string key);

Node \*LL\_rotate(Node \* T);

Node \*RR\_rotate(Node \* T);

Node \*LR\_rotate(Node \* T);

Node \*RL\_rotate(Node \* T);

void InOrder(Node \* node);

};

#endif // \_AVLTree\_H

2、AVLTree.cpp文件

#include "stdafx.h"

#include "AVLTree.h"

#include <algorithm>

Node\* AVLTree::InsAVLTree(Node \*T, Node key)

{

if (!T)

return new Node(key);

if (key.booknum < T->booknum)

T->lchild = InsAVLTree(T->lchild, key);

else if (key.booknum > T->booknum)

T->rchild = InsAVLTree(T->rchild, key);

else

{

T->total += key.total;

T->stillhave += key.stillhave;

return T;

}

T->height = max(GetHeight(T->lchild), GetHeight(T->rchild)) + 1;

int bf = GetBalance(T);

// 左左失衡

if (bf > 1 && GetBalance(T->lchild) > 0)

return LL\_rotate(T);

// 右右失衡

if (bf < -1 && GetBalance(T->rchild) < 0)

return RR\_rotate(T);

// 左右失衡

if (bf > 1 && GetBalance(T->lchild) < 0)

return LR\_rotate(T);

// 右左失衡

if (bf < -1 && GetBalance(T->rchild) > 0)

return RL\_rotate(T);

return T;

}

void AVLTree::Insert(Node key)

{

header->lchild = InsAVLTree(header->lchild, key);

}

int AVLTree::GetHeight(Node \*T)

{

if (!T)

return 0;

return T->height;

}

int AVLTree::GetBalance(Node \* T)

{

if (!T)

return 0;

return GetHeight(T->lchild) - GetHeight(T->rchild);

}

Node\* AVLTree::GetPos\_true(Node \*T, string key)

{

if (T)

{

if (T->booknum == key)

return T;

else if (T->booknum > key)

return GetPos\_true(T->lchild, key);

else

return GetPos\_true(T->rchild, key);

}

else

return NULL;

}

Node\* AVLTree::GetPos(string key)

{

return GetPos\_true(header->lchild, key);

}

Node\* AVLTree::LL\_rotate(Node \* T)

{

Node \*p = T->lchild;

T->lchild = p->rchild;

p->rchild = T;

T->height = max(GetHeight(T->lchild), GetHeight(T->rchild)) + 1;

p->height = max(GetHeight(p->lchild), GetHeight(p->rchild)) + 1;

return p;

}

Node\* AVLTree::RR\_rotate(Node \* T)

{

Node \*p = T->rchild;

T->rchild = p->lchild;

p->lchild = T;

T->height = max(GetHeight(T->lchild), GetHeight(T->rchild)) + 1;

p->height = max(GetHeight(p->lchild), GetHeight(p->rchild)) + 1;

return p;

}

Node\* AVLTree::LR\_rotate(Node \* T)

{

Node \*p = T->lchild;

T->lchild = RR\_rotate(p);

return LL\_rotate(T);

}

Node\* AVLTree::RL\_rotate(Node \* T)

{

Node \*p = T->rchild;

T->rchild = LL\_rotate(p);

return RR\_rotate(T);

}

void AVLTree::InOrder(Node \* node)

{

if (node == nullptr)

return;

InOrder(node->lchild);

cout << node->booknum << ' ' << node->bookname << ' ' << node->author << ' ' << node->stillhave << ' ' << node->total << endl;

InOrder(node->rchild);

}

void AVLTree::print()

{

InOrder(header->lchild);

cout << endl;

}

//插入借书者

void AVLTree::Insertber(string name)

{

borrowerlist[bor].IDnum = name;

borrowerlist[bor].borrowlist = NULL;

bor++;

}

//插入借阅图书的信息

void AVLTree::borrowbook(string IDnum, string booknum, int borrowdate, int returndate)

{

Node \*temp = GetPos(booknum);

if (!GetPos(booknum) || GetPos(booknum)->stillhave<=0)

cout << "此书不存在或已被借完"<<endl;

else //是否有超期未还的书、

{

int i;

bookmsg \*p,\*pre;

for (i = 0; i < bor; i++)

if (borrowerlist[i].IDnum == IDnum)

break;

pre = p = borrowerlist[i].borrowlist;

if (!p)

{

borrowerlist[i].borrowlist = new bookmsg;

borrowerlist[i].borrowlist->booknum = booknum;

borrowerlist[i].borrowlist->borrowdate = borrowdate;

borrowerlist[i].borrowlist->returndate = returndate;

borrowerlist[i].borrowlist->is = false;

borrowerlist[i].borrowlist->next = NULL;

temp->stillhave--;

}

else

{

while (p)

{

p = p->next;

pre = p;

}

pre->next = new bookmsg;

pre->next->booknum = booknum;

pre->next->borrowdate = borrowdate;

pre->next->returndate = returndate;

pre->next->is = false;

pre->next->next = NULL;

temp->stillhave--;

}

}

}

void AVLTree::returnbook(string IDnum, string book)

{

int i;

bookmsg \*p;

for (i = 0; i < bor; i++)

{

if (borrowerlist[i].IDnum == IDnum)

break;

}

if (i == bor)

cout << "ID错误" << endl;

else

{

p = borrowerlist[i].borrowlist;

while (p&&p->booknum!=book)

{

p = p->next;

}

if (p == NULL ||p->is==true)

cout << "没有借过此书或已经归还" << endl;

else

{

p->is = true;

Node \*temp = GetPos(book);

temp->stillhave++;

}

}

}

3、主文件

// 图书馆.cpp: 定义控制台应用程序的入口点。

#include "stdafx.h"

#include <iostream>

#include <fstream>

#include <string>

#include <time.h>

using namespace std;

#include "AVLTree.h"

void printHelp(Node\* p, string ss)

{

if (p == NULL)

return;

ss += " ";

printHelp(p->rchild, ss);

cout << ss << p->booknum << " " << p->stillhave<< " " << p->total<< endl;

printHelp(p->lchild, ss);

}

void print(Node\*p)

{

string ss = "";

printHelp(p, ss);

}

int main()

{

char cmd;

AVLTree avl;

Node temp;

time\_t now;

string bor;

string IDnum;

string booknum;

ifstream in("init\_input.txt");

while (1)

{

in >> temp.booknum;

if (temp.booknum == "#")

break;

in >> temp.bookname >> temp.author >> temp.stillhave >> temp.total;

avl.Insert(temp);

}

in.close();

ifstream inn("borrowmsg.txt");

while (!inn.eof())

{

inn >> IDnum;

avl.Insertber(IDnum);

}

inn.close();

while (1)

{

cout << "N：图书入库 B:借书 R:还书 Q：退出 S：凹入表打印现有书号:";

cin >> cmd;

if (cmd == 'Q')

break;

else if (cmd == 'B')

{

cout << "请输入借阅ID、书号：" << endl;

cin >> IDnum >> booknum ;

avl.borrowbook(IDnum, booknum, (int)time(&now), (int)time(&now)+3600\*24);

}

else if (cmd == 'R')

{

cout << "请输入借阅ID、书号：" << endl;

cin >> IDnum >> booknum;

avl.returnbook(IDnum, booknum);

}

else if (cmd == 'S')

print(avl.header->lchild);

//avl.print();//中序遍历打印

else if (cmd == 'N')

{

int i,n;

ofstream out("init\_input.txt",ios::app);

cout << "输入想要添加的书的种类数量:";

cin >> n;

cout << "输入数据:书号、书名、作者、现存量、库存量";

for (i = 0; i < n; i++)

{

cin >> temp.booknum >> temp.bookname >> temp.author >> temp.stillhave>> temp.total;

out << temp.booknum << ' ' << temp.bookname << ' ' << temp.author << ' ' << temp.stillhave << ' ' << temp.total << ' ' << endl;

avl.Insert(temp);

}

out.close();

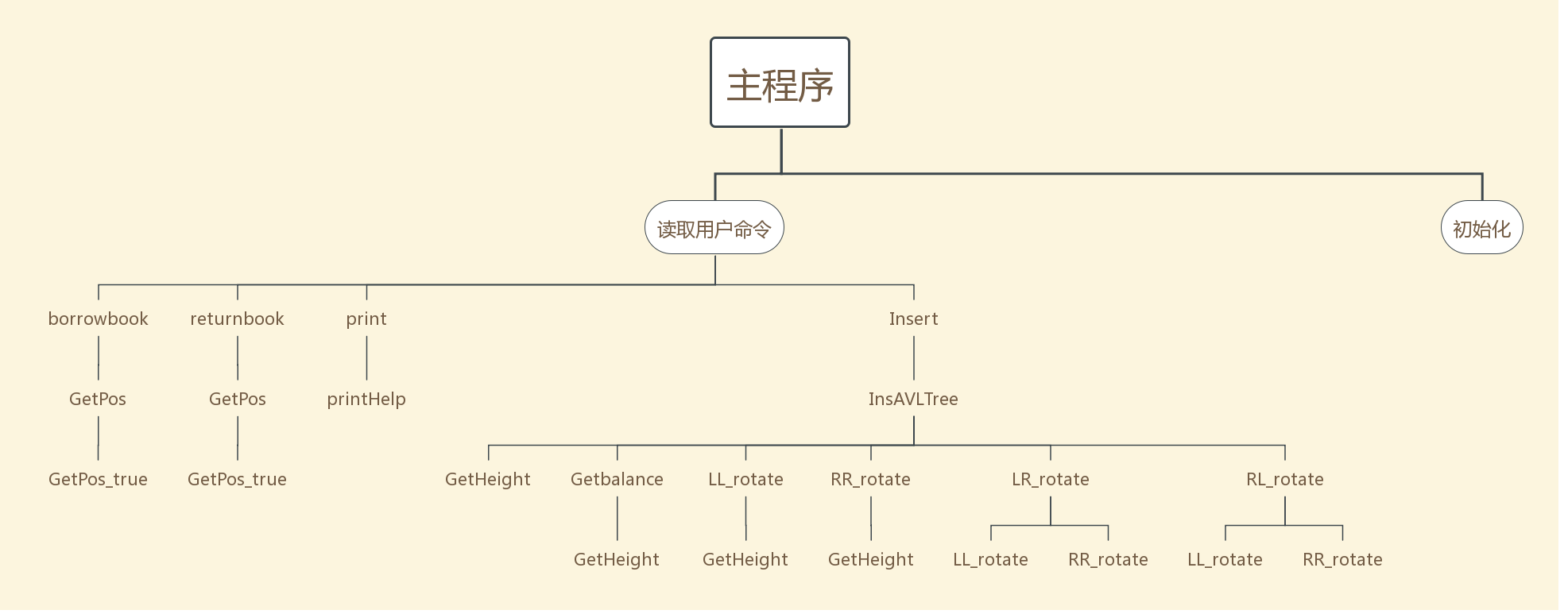
}

}

return 0;

}

函数调用关系图：



四、调试分析

1、此次作业较难，尤其是平衡二叉树的建立较难，采用递归求每个节点高度，再计算平衡因子的值，可以避免繁琐的修改平衡因子。

2、本题中查找算法GetPos的时间复杂度为O（log2n）

3、经验体会：模块化的思想在大型程序中是必需的，不然程序会很难读懂和修改。

五、用户手册

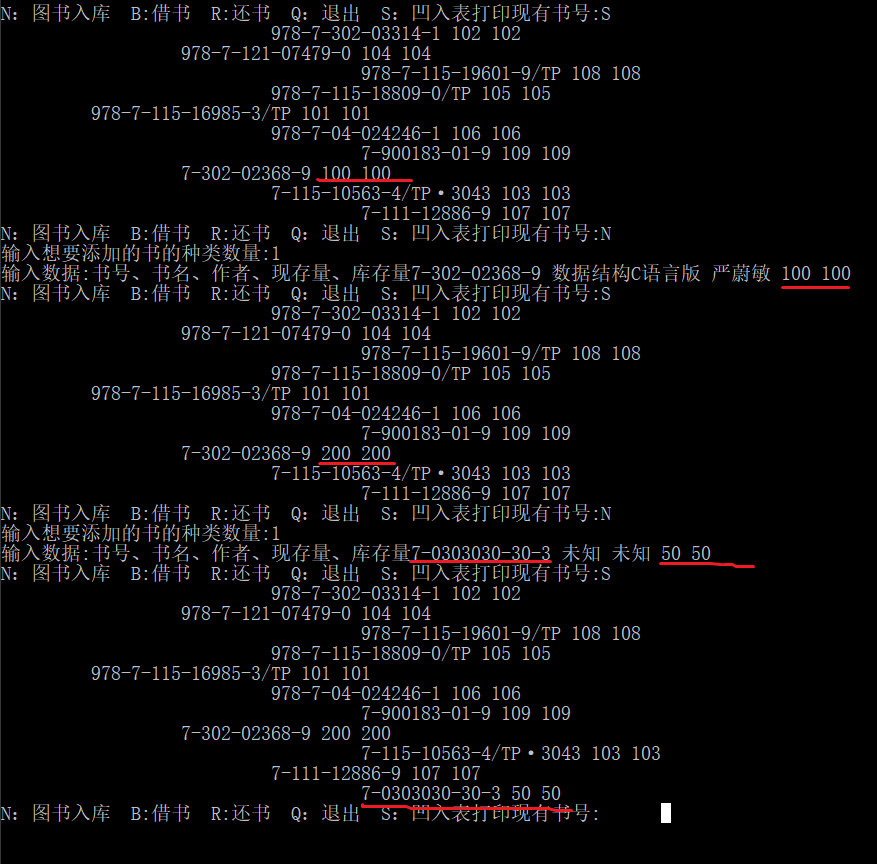
输入菜单命令N：图书入库 B:借书 R:还书 Q：退出 S：凹入表打印现有书号:

输入字符选择功能：

1. 图书入库：先输入想要添加的书的种类数n，种类数n必须大于0；之后输入每本书书号、书名、著者、现存量和总库存量。
2. 借书：输入借书证号、想要借的书的书号。
3. 还书：输入借书证号、想要借的书的书号。
4. 凹入表打印现有书号：输入S，界面会打印出现有的书号。
5. 退出：输入Q，程序退出。

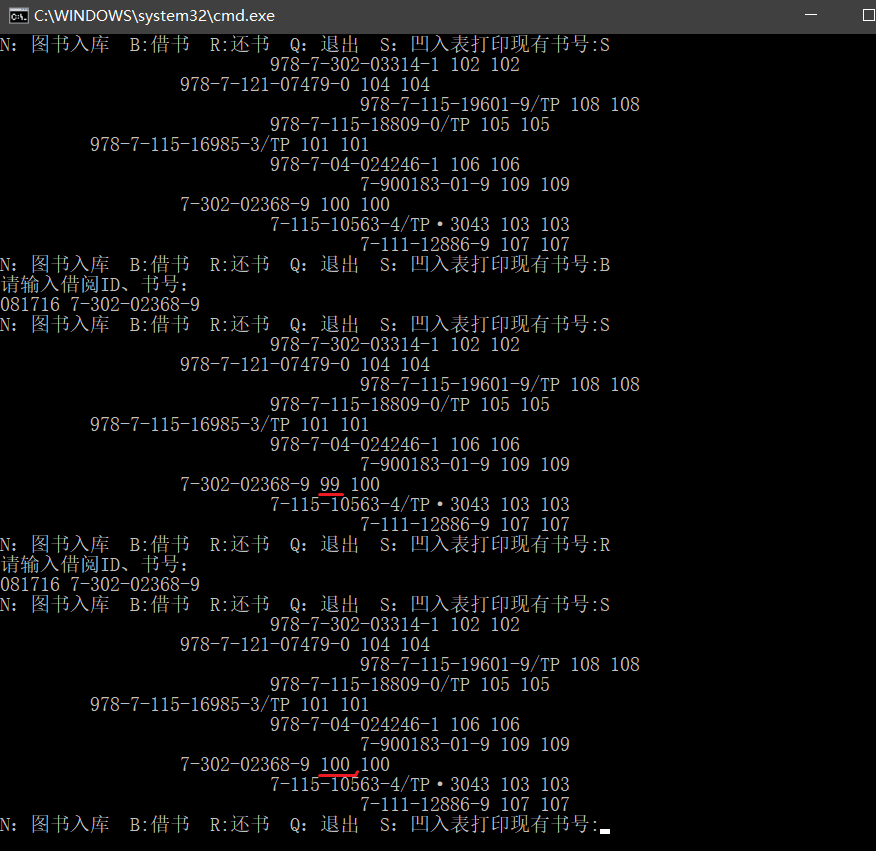
测试结果：

1.图书入库：加入已有的书，相应书的数量会增加；同时也可以加入库存中没有的书

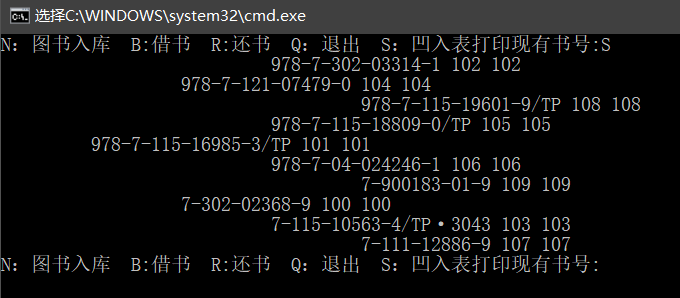


借书还书：输入借阅ID以及书号后，再次打印，可发现相应书的库存量减少了

还书之后，库存量又恢复正常。



凹入表打印：可使用凹入表打印数据，可看到得到的的确是二叉平衡树



七、附录

源程序文件名清单：

图书馆.Cpp //主程序

AVLTree.h //二叉平衡树头文件

AVLTree.c //二叉平衡树源文件