

数据结构课程设计报告

班 级：191162

学 号：20161001979

学生姓名：王彦

指导老师：朱晓莲

日 期：2018年3月

电梯模拟系统

一：问题描述与要求

模拟某校九层教学楼的电梯系统。该楼有一个自动电梯，能在每层停留，其中第一层是大楼的进出层，即是电梯的“本垒层”，电梯“空闲”时，将来到该层候命。

电梯一共有七个状态，即正在开门（Opening）、已开门（Opened）、正在关门（Closing）、已关门（Closed）、等待（Waiting）、移动（Moving）、减速（Decelerate）。

乘客可随机地进出于任何层。对每个人来说，他有一个能容忍的最长等待时间，一旦等候电梯时间过长，他将放弃。

模拟时钟从0开始，时间单位为0.1秒。人和电梯的各种动作均要消耗一定的时间单位（简记为t），比如：

有人进出时，电梯每隔40t测试一次，若无人进出，则关门；

关门和开门各需要20t；

每个人进出电梯均需要25t；

电梯加速需要15t；

上升时，每一层需要51t，减速需要14t；

下降时，每一层需要61t，减速需要23t；

如果电梯在某层静止时间超过300t，则驶回1层候命。

电梯调度规则：

1）就近原则：电梯的主要调度策略是首先响应沿当前行进方向上最近端的请求直到满足最远端请求。若该方向上无请求时，就改变移动方向；

2）在就近原则无法满足的情况下，首先满足更高层的请求；

3）电梯的最大承载人数为13人，电梯人数达到13人后，在有人出电梯之前，不接受进入电梯的请求；

4）乘客上下电梯时先出后进。进电梯时乘客是按发出乘坐请求的顺序依次进入，每次只能进入一人且每个人花费的时间都为25t；

5）电梯在关门期间（电梯离开之前）所在层提出请求的乘客同样允许进入。

要求：

按时序显示系统状态的变化过程，即发生的全部人和电梯的动作序列。

扩展要求：

实现电梯模拟的可视化界面。用动画显示电梯的升降，人进出电梯。设计有下列对象：电梯、人、电梯控制板及其上各种按钮、定时器等。

二：需求分析

电梯调度问题实质上是解决在复杂的实际情况中电梯的决策问题，程序的主要功能是对某校九层教学楼的电梯的运行系统进行模拟，程序要求要想实际生活中一样，可以实时的添加乘客，即乘客们请求电梯的时间不是固定的，是随机生成的，输入的数据即是乘客的信息（排序，起始楼层，终点楼层，最大等待时间），输出电梯的实时状态（在某时刻的状态既所在楼层）

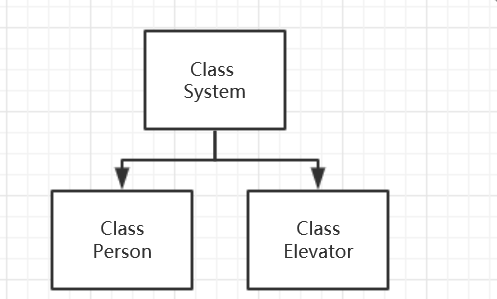
三：设计

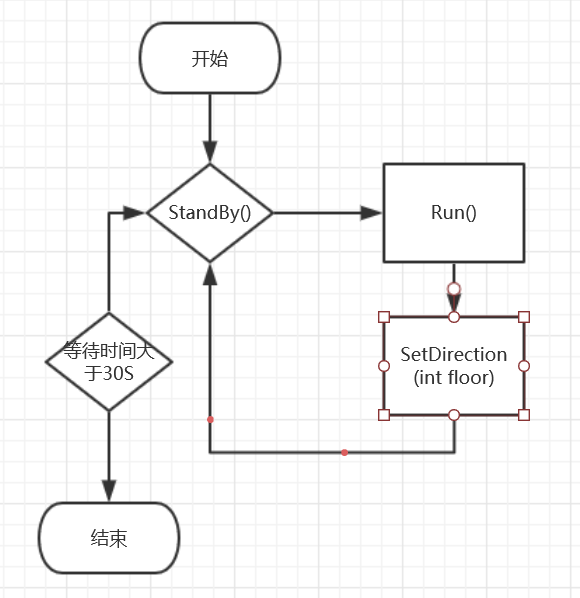
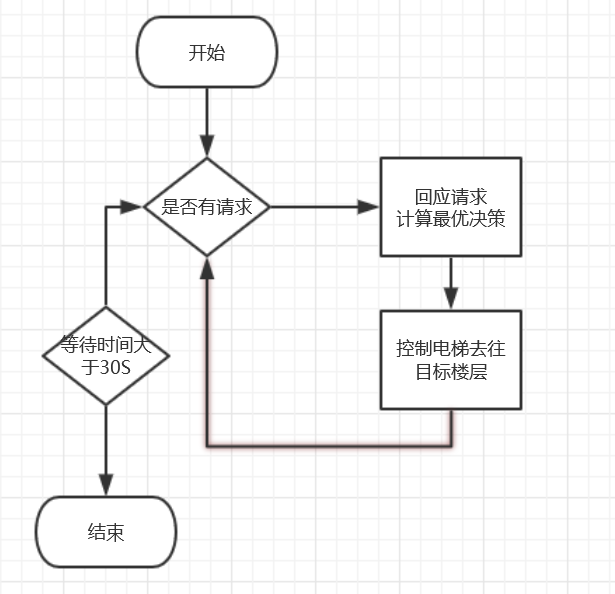
**3.1：设计思想**

所用的数据结构采用链表（队列），链表没有头节点，下标从1开始，对教学楼的所有乘客进行储存，一共有两个链表，一个链表储存所有在电梯内的人，一个链表储存所有在电梯外的人，采用队列的思想，满足先到的人先处理请求。而乘客的起始楼层和目的楼层则作为乘客自身的数据成员定义在Person类中，这样做在时间上就会有较大消耗，因为当电梯运行到每一层判断有没有乘客到达目的地时候，需要遍历整个链表，对每个节点进行判断，效率很低，但是这样很方便控制处理，所有数据在同一个结构中储存着，可以整体进行管理运算，所以采用这种数据结构。对于时间事件我采用的是Windows.h中的延时函数，将系统延时后输出语句，用于体现电梯的动态过程。

程序构架如下：

程序整体架构如下：

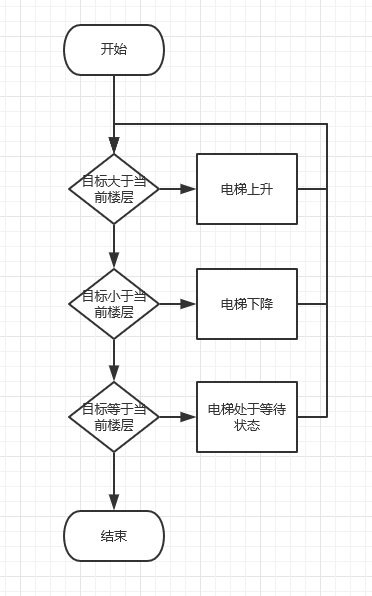




StandBy()函数实时判断当前是否有乘客请求，如果有乘客请求就转入Run()函数进行处理，如果没有就递归调用自身实时检查是否有请求，如果等待时间大于30秒就返回一楼结束程序。以及Initialize()用于系统的初始化(获得起始时间)，bool GoOut()函数用于判断当前楼层是否有人出去，bool GoIn()函数用于判断当前楼层是否有人进入，还有void Out()用于真正上将处于电梯中(链表中)的人删除出去，void In()用于真正上将电梯外的人删除并且加入到电梯中去。Arrive()通过调用bool GoOut()和bool GoIn()实现判断该楼层是否有人请求，总体上实现到达楼层判断，此外，还有beyondToleration()函数用于判断是否有人到达忍耐时间。下面是函数的具体实现。

StandBy()函数中作为等待命令的函数，每一秒钟调用一次用于检测是否有请求出现，在函数中使用(\_kbhit())函数获取键盘事件，用来实现实时加人的目的，每次用户按下键盘便随机生成几个人，并将他们储存在外链表中等待处理(转入Run()函数)，另外，通过调用 BeyondToleration()函数检测是个有人等不及了，设置Wait变量用于检测是否到了最大等待时间，如果到达了最大等待时间就回到本垒层结束程序，如果没有到就继续递归或者处理事件。

SetDirection(int floor)函数做简单的上下楼层处理，函数体内进行判断，如果当前楼层小于目标楼层就上升直到到达目标楼层，如果当前楼层大于目标楼层就下降到目标楼层，并在控制台输出状态语句，流程图如下：



Bool GoIn()函数判断是否有人要进入电梯，遍历整个数组查看每个节点(乘客)的起始楼层是否等于当前楼层，如果等于则返回true反之返回false。当然在函数中判断电梯是否超载。

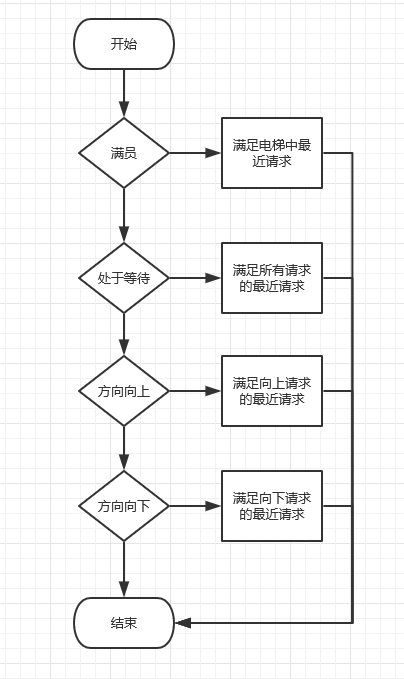
Bool GoOut()函数同理判断是否有人在当前楼层出去，同样采取遍历的方法(这里效率比较低下有待改进)

Out()函数对应上面的GoOut()函数其真正将能出去的人在里链表中删除，以及电梯状态的改变，在程序中的用法是if(GoOut()==true) Out();

In()函数同上将真的将能进去的人在链表中删除，以及电梯状态的改变，在程序中的用法是if(GoIn()==true) In();不一样的是这里要判断电梯是否超载，如果超载就停止遍历链表转而处理电梯中已经有的13人。

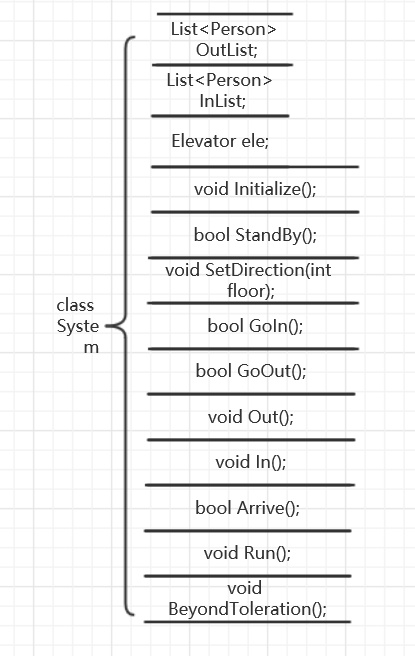
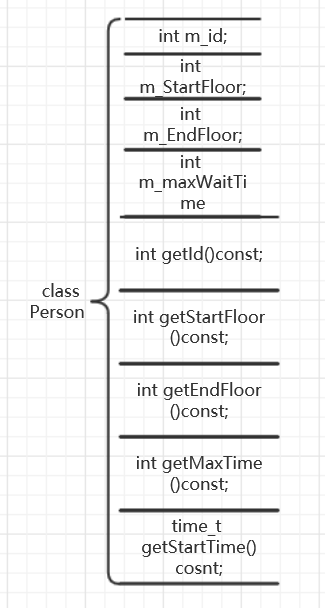
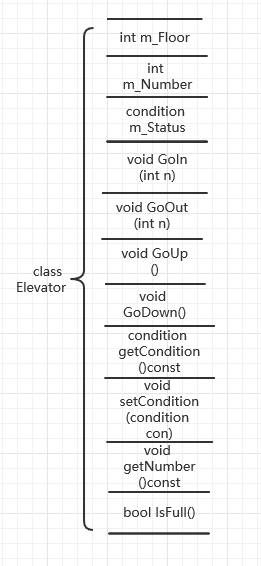
Arrive()函数通过调用Goin() GoOut()函数判断当前楼层是否有人要到达，并即时输出减速制动语句。

Run()函数是最主要的电梯调度函数，分为几种情况，当电梯是超载状态的时候不回应电梯外的人的请求，只处理电梯中的人的请求，计算出电梯里请求的最近楼层，然后将楼层传入SetDirection()让电梯到达，如果电梯处于等待请求状态就查看全楼的请求包括电梯外面和电梯里面，找到最近的目标楼层然后到达，而如果电梯此时是有方向的，就率先满足同方向的最近请求，例如电梯方向向上，就只查看向上的请求并在向上的请求中选出最近的请求，方向向下同理。流程图如下：



**3.2设计表示**

电梯类的数据成员有当前所在楼层(int m\_Floor)当前载重人数(int m\_Number)和当前状态m\_Status(枚举类型Opening, Opened, Closing, Closed, Waiting, Up, Down, Decelerate)成员函数除必须的构造函数，析构函数，复制构造函数外还有接口函数int getfloor()const,int getNumber()const;void setCondition(condition con),condition getCondition()const;用于获取电梯的当前楼层，电梯的当前状态，电梯的人数还有设置电梯的状态。此外对电梯行为进行操作的GoIn(int n),GoOut(int n),GoUp(),GoDown(),分别将电梯中的人数增加或者减少，电梯所在的楼层增加或者减少。还有判断电梯时候满员的函数Bool IsFull();类框架如下：



**3.3详细设计：**

伪代码：

Bool System::StandBy(){

If(键盘事件){

随机生成相应的人数；

将乘客加入外链表；

}

//检测是否有人等不及了

If(电梯处于等待状态) {记录等待时间变量增加}

If(电梯处于等待状态而且等待时间大于30秒){

返回一楼；

结束；}

Else 转入电梯调度函数；

}

Void System::SetDirection(int floor){

//检测是否有人等不及了

If(当前楼层大于目标楼层){

电梯的方向设置为向下

While(电梯楼层不等于目标楼层){

电梯所在楼层减一；}

到达目标楼层，处理进人出人；

}

Else if(当前楼层小于目标楼层){

电梯的方向设置向下

While(电梯楼层不等于目标楼层){

电梯所在楼层加一;}

到达目标楼层，处理进人出人；

}

Else if(电梯楼层处于当前楼层){

电梯楼层处于等待状态

处理进人出人事件；

}

Void System::Run(){

If(电梯超员){

If(电梯中有人){

找到最近请求；

转到运行函数；}

}

Else if(电梯处于等待状态){

If(电梯中有人){

找到最近请求，记为Near1；

}

If(电梯外有人){

找到最近请求，记为Near2；

}

判断Near1和Near2的大小，取更小的值为near；

SetDirection(near);

}

Else if(电梯处于上升状态){

If(电梯中有人){

找到最近请求，记为Near1；

}

If(电梯外有人){

找到同方向上的最近请求，记为Near2；

}

判断Near1和Near2的大小，取更小值为near；

SetDirection(near);

}

Else if(电梯处于下降状态){

If(电梯中有人){

找到最近请求，记为Near1；

}

If(电梯外有人){

找到同方向上的最近请求，记为Near2；

}

判断Near1和Near2的大小，取更小值为near；

SetDirection(near);

}

四：调试分析：

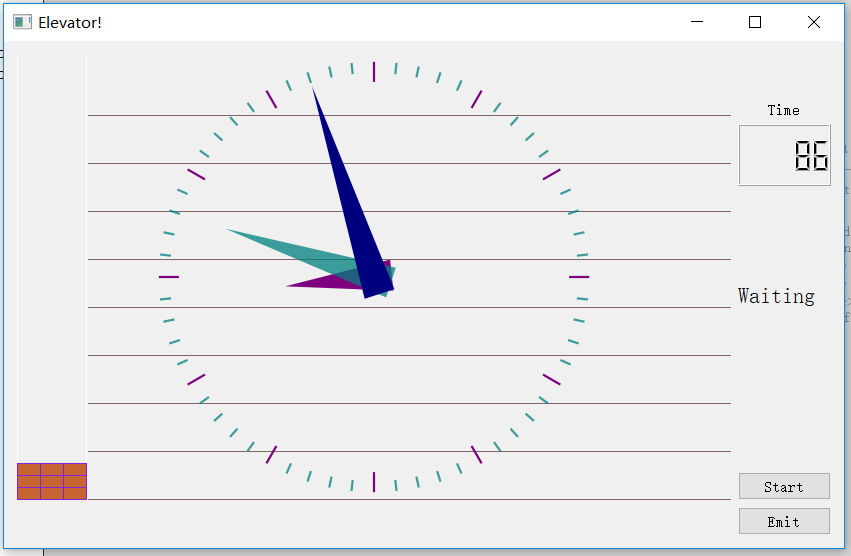
**4.1遇到的问题及其改进**

调试分析工程中遭遇到了很多问题，程序也在改进Debug中不断进步，从最初的version 1.0到现在的version 3.0程序有较大改进，从最初的仅仅符合最近调度原则到现在的符合同方向的就近原则，解决办法是在调度函数Run()中加入针对电梯状态的判断以及乘客方向的检查，

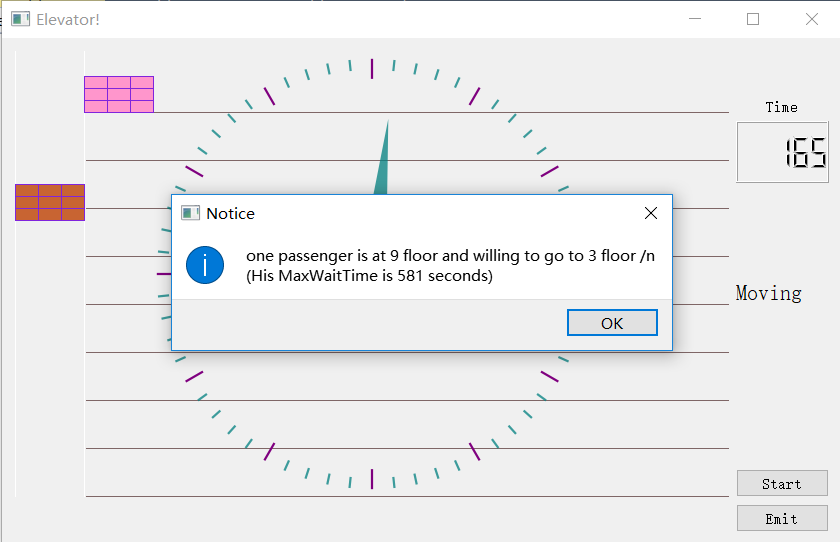
其次更新了对于电梯超载问题的解决，Version1.0不具备对电梯超员后的处理能力，更新后的程序在电梯进人函数中加入电梯人数的判断逻辑，如果电梯满员就只响应电梯中的人的请求，对于电梯外的人的请求不予理睬。实现对于电梯超载问题的解决。

其次实现了乘客的实时加入，最开始的版本1.0只能在程序初始化过程中随机生成人，其随机具备一个开始等待时间，当时间线到达其等待时间后将其加入等待列表开始处理其相应。而在最新版本中在StandBy()函数中加入获取键盘事件的逻辑，可以实时获取用户的输入，当用户输入一个数字后，程序自动随机生成相应的人数。

同时加入可视化设计，Qt界面如下：



左面3\*3棕色方块表示电梯，旁边九条横线表示楼层，主体部分一个时钟表示实时时间，右上方的数字同样表示时间，不过其以50毫秒为单位递增作为时间事件参与客户区的重绘，下面有表示电梯当前状态的Label，下方两个按钮分别用于系统开始运作和乘客的生产



按下Emit按钮随机生成一名乘客，并弹窗提示用户其所在楼层及其目标楼层，同时在客户区生成粉色3\*3方块表示乘客，当乘客进出电梯时同样有语句提示。

信号与槽函数

public slots:

void getstart();

void getappear();

void getstatue();

signals:

void time (const int);

void statue();

void appear(const int);

绘图事件与时间事件：

void paintEvent(QPaintEvent\*)Q\_DECL\_OVERRIDE;

void timerEvent(QTimerEvent\*)Q\_DECL\_OVERRIDE;

实现客户区的重绘以及时间信号的更新。

**4.2算法时间空间复杂度分析：**

若乘客的数量为n，则Run()函数的时间复杂度为O(n),空间复杂度为O(1);

SetDirection(int floor)函数时间复杂度为O(n),空间复杂度为O(1);bool GoIn()，bool GoOut()函数时间复杂度为O(n),空间复杂度为O(1); void In()，void Out()函数时间复杂度为O(n),空间复杂度为O(1);bool Arrive()函数的时间复杂度为O(1),空间复杂度为O(1);bool StandBy()函数的空间复杂度为O(n),时间复杂度为O(n).

**4.3算法改进：**

从上面可以看出算法的效率不高，因为当初开始设计电梯调度系统时认为其不会涉及到大量数据，所以并没有思考算法的优化，现在想来，还有很大的改进空间。

第一：类似于开散列结构，对每个楼层设计一个链表，乘客发出请求时将其加入对应楼层的链表，这样来在查找对应楼层的乘客的效率会大幅度提高。构造函数大体如下：

for(int i = 0; i < 10; ++i)

{

OutList[i] = new Node(0);

OutList[i]->link = nullptr;

}

析构函数如下：

for(int i = 0; i < 10; ++i)

{

t = OutList[i];

while(t->link != nullptr)

{

Node\* r = t->link;

t->link = r->link;

delete r;

}

delete t;

}

其二：可以在搜索函数上做改进，采用二分法查找使搜索的时间复杂度降低到O(logn)而不是现在的遍历的O(n);

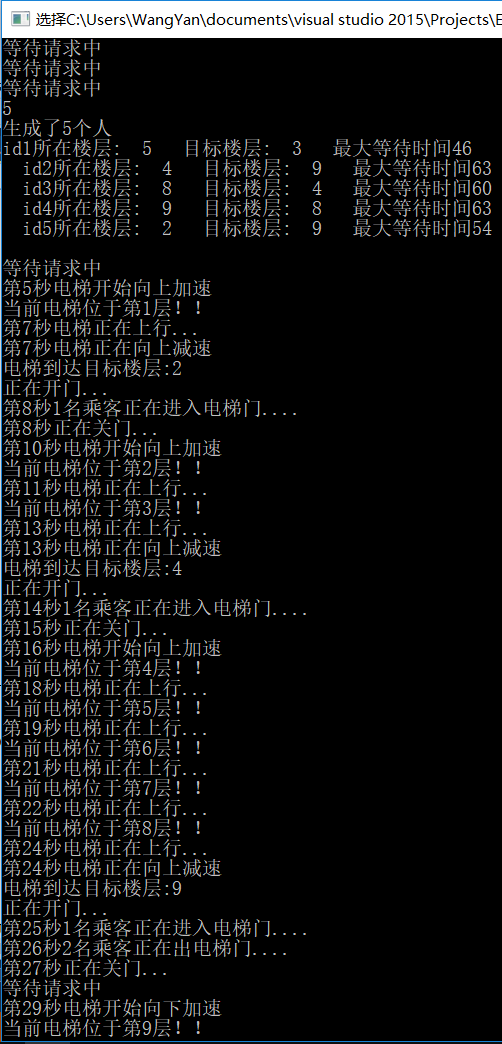
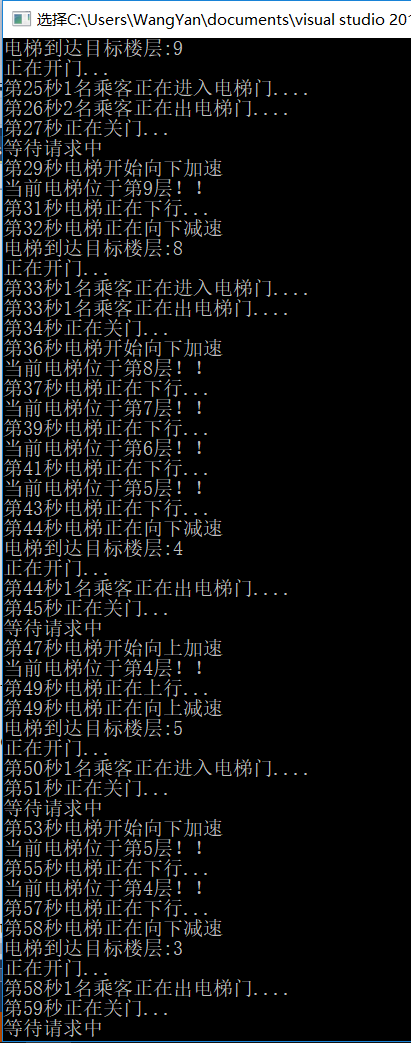
五：用户手册README

运行程序，电梯处于空闲等待状态，用户从键盘输入一个数字，程序自动生成对应个数字的相应人数，其起始楼层与目标楼层随机，电梯开始处理请求，用户可以继续从键盘输入数据，程序将相应并再次生成乘客，直到运送完所有的乘客并且电梯空闲等待时间大于30秒程序结束。

六：测试数据及测试结果。

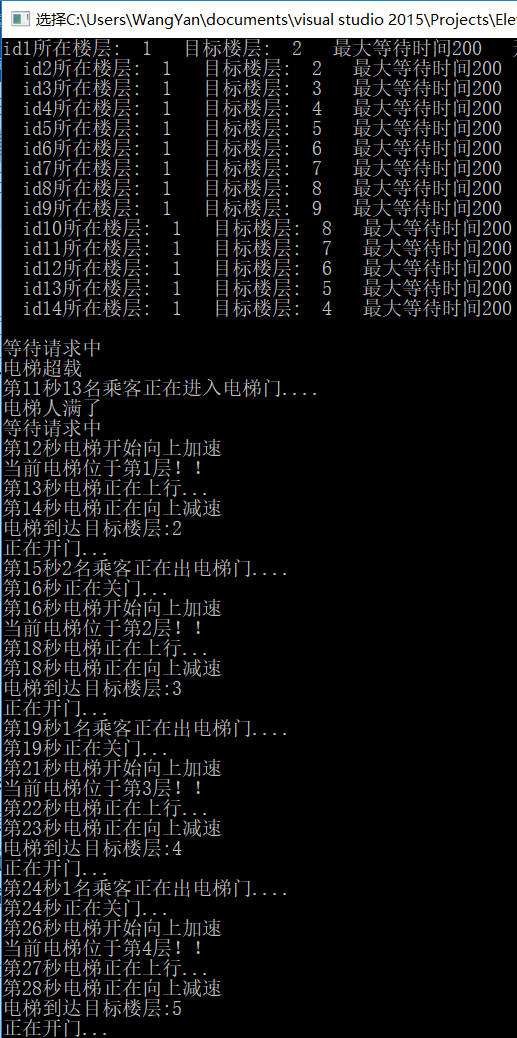
1.测试人数5，测试电梯的调度函数是否满足就近原则和同方向就近原则，调度是否合理。生成5个人，分别是5-3 4-9 8-4 9-8 2-9(起始楼层-目标楼层)若程序正确则输出2-4-9-8-4-5-3

运行结果如下：

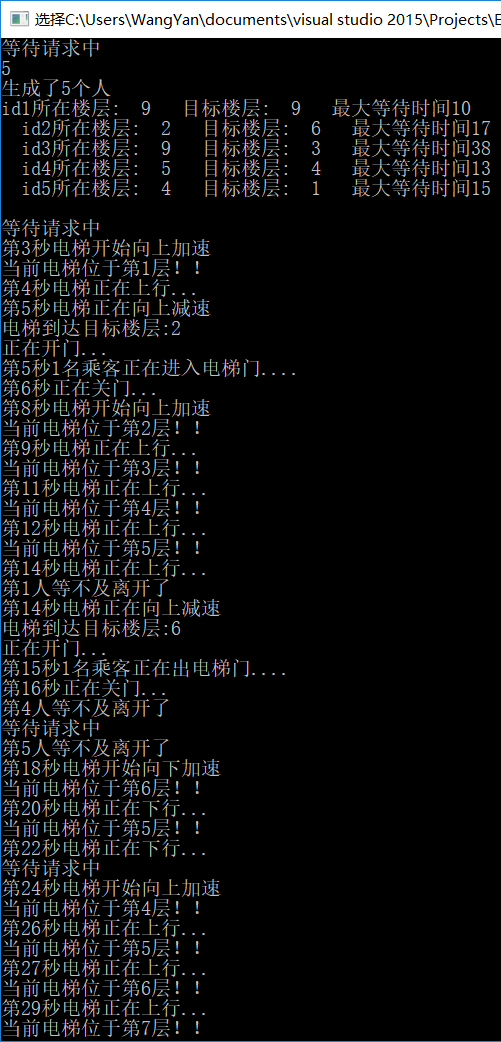
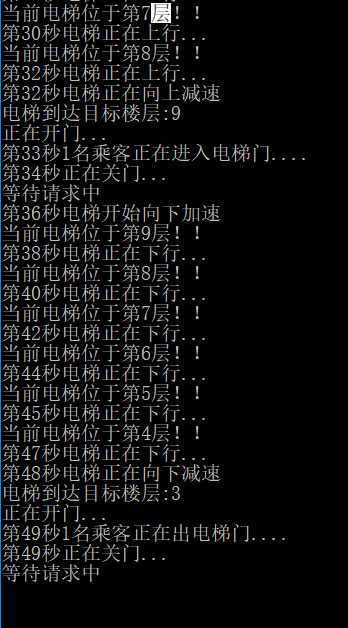
当前状态通过。

2.假设有14人同时从1楼发出请求，目的检测电梯对于超员状态的处理。



电梯在一楼仅只进入13人，先满足这13个乘客的请求，当电梯中人数不足13人时再响应第14个人的请求。当前状态通过。

3.测试人数5人，将其最大等待时间调整为较小值，查看电梯对于等待超时事件的响应。

电梯运行第15时第一位乘客等待不急离开了，其等待时间仅有十秒，在第16秒时第4第5为乘客相继离开，其等待时间分别为13与15秒，数据在合理范围内，但是可以看出有部分延时，即等待时间为10秒时，当运行时间等于10秒时乘客不会立即离开，而是会有一两秒的差距，是由于函数调用时期导致的，用于判断是否到达忍耐时间的函数并不会(也不能)每时每刻都调用，其只在部分阶段调用所以不能即时判断出乘客是否离开，这是有改进空间的。

七：源程序清单：

System.cpp//电梯调度

Main.cpp//主函数

Elevator.cpp//电梯行为函数实现

System.h//调度系统成员函数声明及数据成员定义

Person.h//乘客类定义及其实现

List.h//链表类

Elevator.h//电梯成员函数声明及其数据成员定义

图书管理系统

一：问题描述与要求

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。

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。但由于上述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（采用2-3-4树）对书号建立索引，以获得高效率。

（3）系统应实现的主要功能如下：

采编入库：新购入的图书经分类和确定书号后添加到图书账目中，如果该书已存在，则只将总库存量增加；

清除库存：某种书已无保留价值，把它从图书账目中注销；

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；

归还：注销对借阅者的登记，改变该书的现存量；逾期归还的，给出罚款数额；

显示：以凹入表的形式显示B树。该功能是为了调试和维护的目的而设置的。

扩展要求：

（1）增加列出某著者的全部著作的操作，思考如何提高该操作的效率。

（2）增加列出某种书状态的操作。状态信息除了包括这本书记录的全部信息外，还包括最早到期（包括已逾期）的借阅者证号，日期可用整数实现。

（3）增加预约借书功能。

二：需求分析

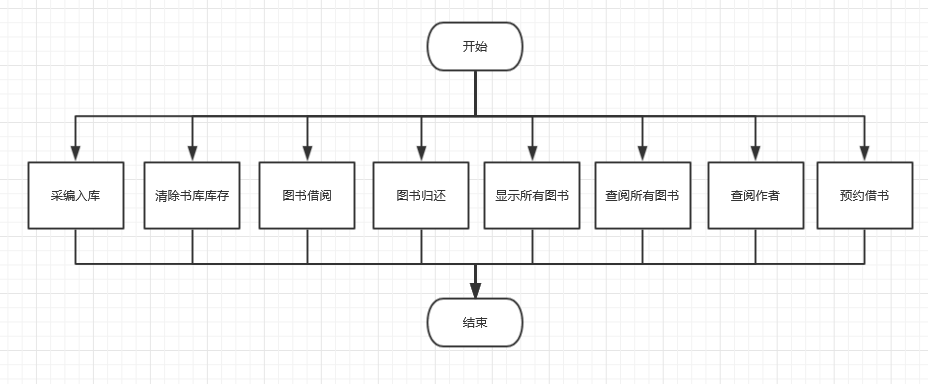
问题需要解决图书馆借书还书等相关问题，程序能够实现采编入库，清除库存，借阅，归还，显示和查询功能，用户可以根据自己的需求选择适合的功能程序执行相应的功能。

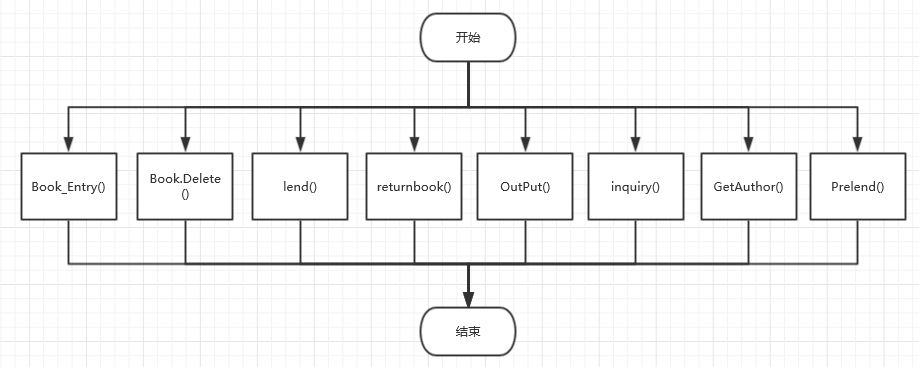
三：设计

**3.1设计思想**

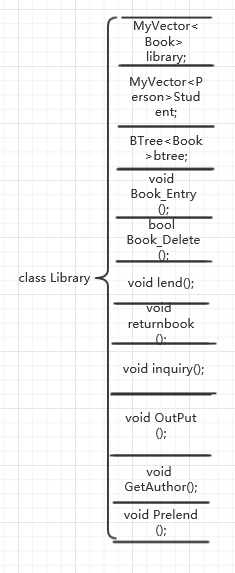
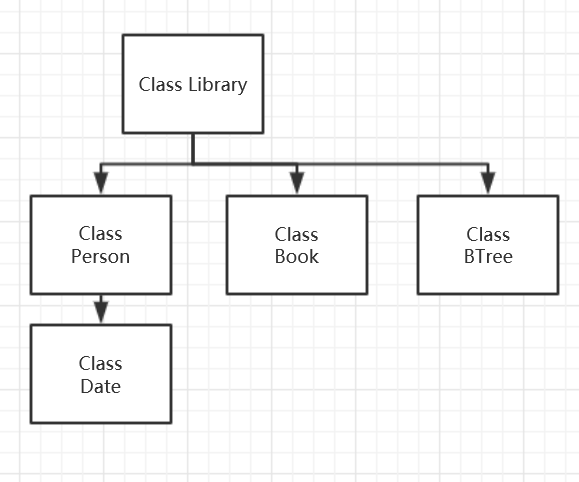
全部数据可以都在内存存放。但由于上述基本业务活动都是通过书号（即关键字）进行的，所以要用B树（采用2-3-4树）对书号建立索引，以获得高效率。程序采用4阶B树作为索引结构，采用自定义容器类MyVector作为存储结构，图书馆结构中用一个MyVector存放所有的图书，查找图书时采用B树查找，找到后返回该图书在容器中的下标索引，MyVector采用下标直接访问数据，可以提高查找的效率，而且操作方便合理，故采用上述数据结构。

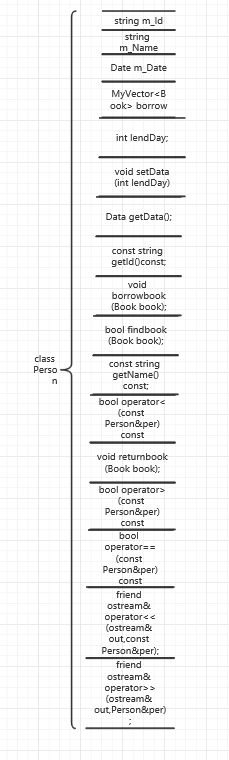
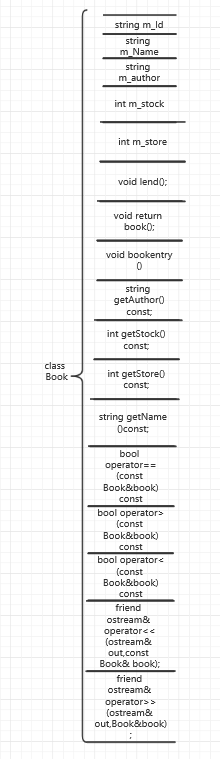
程序框架如下：





类框架如下：



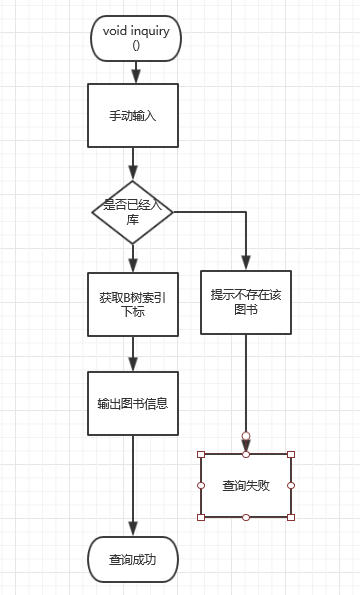
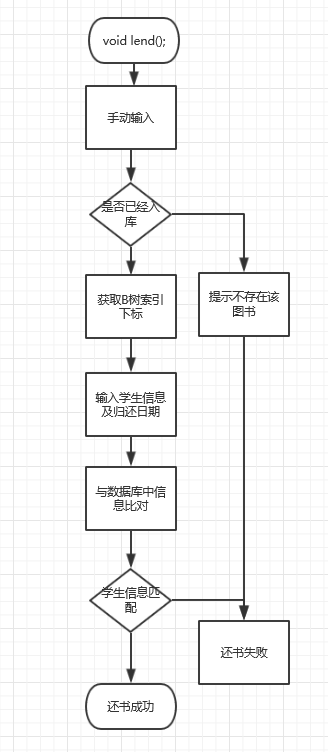
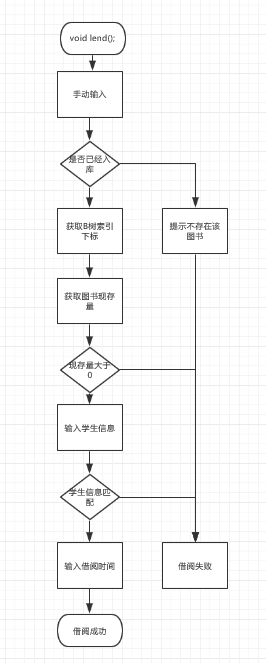
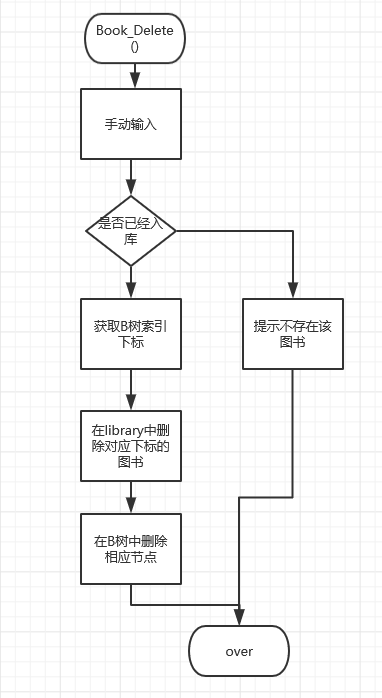
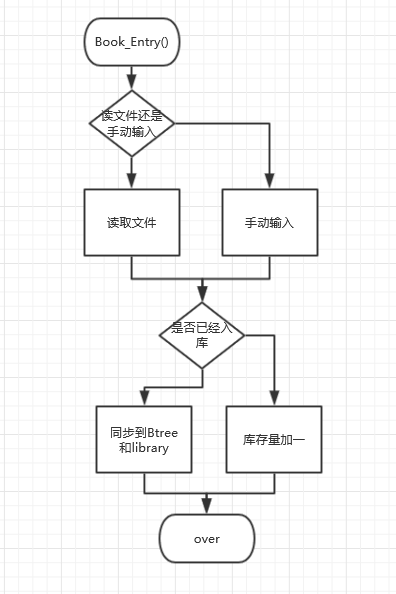
 

**3.2设计表示：**

Person类中包含数据成员编号m\_Id,姓名m\_Name,日期m\_Data,所借的图书borrow,还有借阅日期lendDay，除了获取私有成员变量的接口函数外，还重载了比较函数用于对借阅人进行查找排序(以m\_Id为关键字)，同时重载输入输出符方便表示与录入，同样Book类中也有对于大小比较函数，输入输出符号的重载，目的相同，此外Book类还具有数据成员m\_stock现存量以及m\_store 总库存量，除接口函数外还拥有void returnbook();和void bookentry();函数，分别简单实现现存量和总库存量加一，同样是作为Library类的接口方便书籍入库以及还书的实现。

Class Library中具有数据成员MyVector<Book>library储存着所有的图书，MyVector<Person>Student储存着所有的学生信息，还有BTree<Book>btree对所有书籍建立索引，成员函数有Book\_Entry(),bool Book\_Delete();void lend();void returnbook();void inquiry();

Void OutPut();void GetAuthor();void Prelend();具体实现



Book\_Entry()图书入库函数，用户可以选择读取文件或者手动输入，完毕后根据输入的书号使用Btree进行索引，返回关键字下标，如果已经存在，则输出提示无需入库，仅将库存加一处理，如果不存在，就将其加入到library中，同时在Btree中添加相同节点用于索引。

bool Book\_Delete()删除图书同样，让用户输入一本书的相关信息，先使用B树进行查找，根据返回的下标判断是否已经存在(不存在为-1，存在为>-1)，在library中删除对应该下标的图书，同时将B树中对应节点删除，如果没有找到就提示用户查找失败。

void lend()借阅图书，先让用户输入一本书的相关信息，先使用B树进行查找，如果找到就返回一个下标，先查看该下标对应图书的现存量，如果现存量大于0，则表示有这个书可以借阅，则要求用户输入其学号姓名，与数据库中信息比对如果一致则同意借阅，要求用户输入借阅日期，借阅成功，反之借阅失败。

void returnbook()还书函数，先让用户输入一本书的相关信息，先使用B树进行查找，如果找到就返回一个下标，让该下标对应的图书的库存量增加一，同时要求学生输入个人信息，在学生信息中查找，如果找到就在该学生的所借阅图书信息中删除该图书，还书完成。

void inquiry()查询函数，先让用户输入一本书的相关信息，先使用B树进行查找，如果找到就返回一个下标，输出library中对应下标元素的所有信息。

**拓展功能**：

查询作者，用户输入一个姓名，在library中遍历作者姓名进行比对，如果找到符合的就输出图书的相关信息，否则查询失败，该功能没有用到B树，效率比较低。

预约借书，先让用户输入一本书的相关信息，先使用B树进行查找，如果没有找到就新建立一本图书，将这本图书添加到library中并同步到B树中，提示用户预约成功。

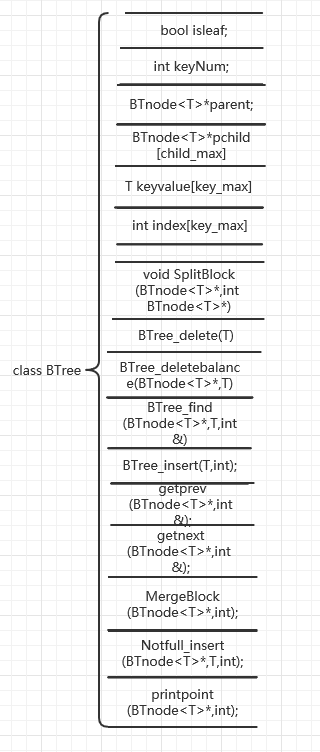
输出函数，通过调用B树的凹入打印函数将全部图书信息以凹入表形式打印出来，方便用户查看。

**3.3详细设计**

B树节点共有6个数据成员，分别是标记其是否为叶子的成员bool isleaf和记录其关键码的个数的值int KeyNum，以及其父亲节点指针和子女节点指针BTnode\* parent;BTnode\* pchild[child\_max],和关键字数组template<T> T keyvalue[key\_max]和其对应的下标值数组int index[key\_max];

函数成员如下

用于节点分裂的void SplitBlock(BTnode<T>\* node\_parent,int child\_index,BTnode<T>\*node\_child)；用于插入关键字的void Notfull\_insert(BTnode<T>\* node, T value,int n), bool BTree\_insert(T value,int n)用于节点合并的void MergeBlock(BTnode<T>\* node\_parent, int child\_index)，用于节点替换的T getprev(BTnode<T>\* node,int& n) T getnext(BTnode<T>\* node,int&n)用于查找的BTnode<T>\* BTree\_find(BTnode<T>\* node, T value,int&n)以及凹入打印函数，void printpoint(BTnode<T>\* node, int count)和删除函数void BTree\_deletebalance(BTnode<T>\* node, T value)，类结构示意图如下：



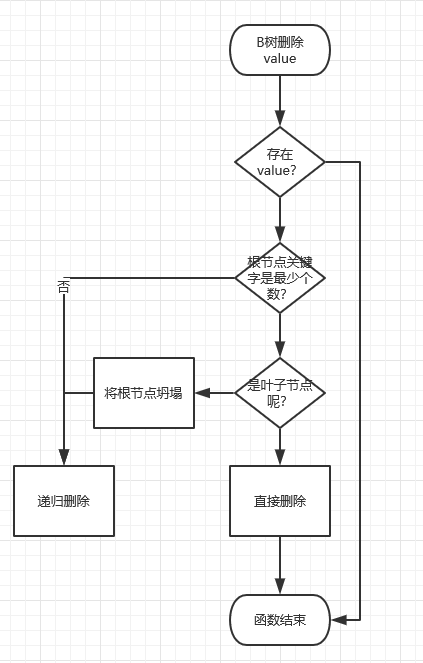
B树插入分为两种情况，未满插入和已满插入，当节点已满时，通过SplitBlock函数将节点分块变成两个未满的块，这样问题就变成了在一个未满的块添加关键字的过程，未满插入函数先判断该节点是否是叶子节点，如果是叶子节点就直接插入没有什么影响，如果不是叶子节点就在节点的关键字中找到适合的位置(比它小的在左边，比它大的在右边)，然后判断该节点的关键字个数是否超过限制，如果超过限制就及时分裂，如果没有超过就继续递归，往下层中找适合位置查找，直到叶子节点为止。

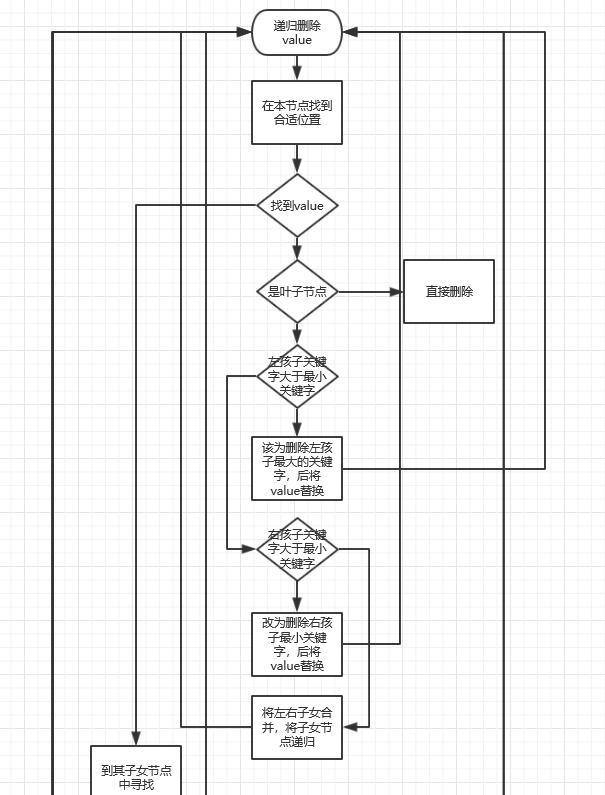
SplitBlock函数节点分裂函数，代入参数为父节点，孩子节点及其对应下标，函数将子节点分裂为两个节点，并将子节点中间的关键字提到父节点上，完成节点的分裂

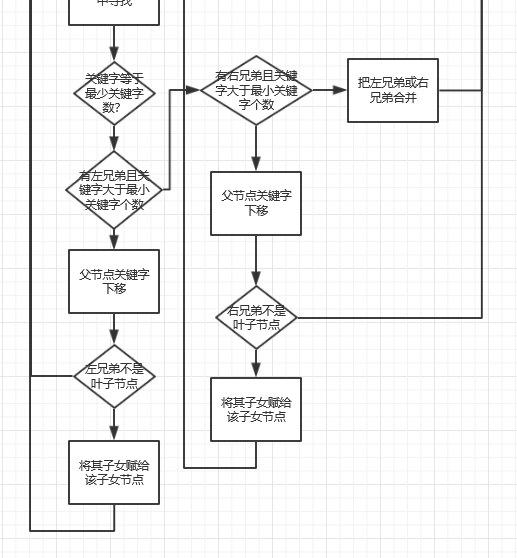
 BTnode<T>\* BTree\_find(BTnode<T>\* node,T value)函数同样是递归的过程，现在本节点中找想寻找的关键字，如果找到就寻找结束，返回元素下标，如果没有找到，就在孩子节点中递归去找，直到递归到叶子节点，函数结束，返回nullptr表示没找到。凹入打印函数同样递归的思想，带入参数count表示递归的深度，先判断是否是叶子节点，如果不是就要到子节点中递归打印，然后将本节点信息输出。通过递归将B树输出。

合并函数将两个块未满的块合成一个块，参数为父节点和一个子节点的下标，将该下标对应的两个孩子节点合并为一个，首先将父亲节点对应的关键字下移，在将右孩子的关键字赋给左孩子，同时将右孩子的子女赋给左孩子，删除右孩子，保留左孩子，这样两个块就合成了一个块。T getprev(BTnode<T>\* node)  T getnext(BTnode<T>\* node)  是用于关键字替换的函数，作用分别是将左兄弟中最大的关键字返回出来，将右兄弟中最小的关键字返回出来，用于函数的关键字替换。

删除函数是B树的精髓所在，整体采用递归的思想，先从根节点开始一层一层向下层递归，直到遇到根节点为止，在整个过程中始终保持B树下层的稳定(节点关键字大于最小关键字)防止树的坍塌影响上层的结构，具体做法如下，先将根节点带入删除函数，此时判断根节点是否只有一个关键字并且其子女的关键字等于最小限制的关键字个数，如果这样那么在删除时很有可能会引起树的坍塌，所以将根节点率先塌下来保持树的稳定，如果不是这样，那么就在根节点找要删除的value，如果找到而且是叶子节点则到达递归终点函数结束，如果不是叶子节点就要查看它的左右子女，如果其子女关键字个数均等于最小限制的关键字个数还是会导致坍塌，就将左右子女合并，并递归到其孩子节点中，如果其任一个子女的关键字大于最少关键字就去该子女中合适的关键字current取代当前关键字，改为删除current然后将value改为current，如果在节点中没有找到要删除的关键字，就到该节点的孩子节点中去找，如果孩子节点中同样只有最少的关键字，就将该节点坍塌下来与孩子节点合并，取左兄弟或者右兄弟的合适关键字赋值给父节点，若兄弟节点不是叶子节点就还要把子女赋值给那个孩子节点，再将孩子节点作为参数进行下一次递归。由于函数逻辑复杂语言表述很难清晰易懂，那么把流程图画出来。







四：调试分析

**4.1遇到的问题及其改进**

这道题的难点在于B树的使用，由于先前借鉴网上的代码，对其思想理解还不透彻，通过自己的调试分析，已经完全理解了B树的具体实现，比如下图要删除节点2，算法运行如下，先从根节点开始检测，查看关键字是否在根节点中，发现不在，而且关键字比根节点的keyvalue要小，所以开始检测根节点的左边的孩子节点,发现这个孩子节点只有一个关键子字，这样在删除时很容易使B树坍塌，所以要向其兄弟节点借，发现其没有左兄弟，但是有右兄弟，则先将父节点的关键字下移，再将右兄弟节点中最小的关键字赋值给父节点对应位置，同时需要把其最左边的子节点赋值给这个孩子节点，再将孩子节点作为参数进行递归删除，在这个节点中查找关键字，找到了，但是它不是叶子节点不能直接删除，发现其孩子节点的关键字都为1，这样删除同样会导致不平衡的坍塌，所以将该节点和其孩子节点合并为一个节点，再次执行递归删除，这次，在该节点中发现了关键字，而且它变成了叶子节点，所以直接删除，递归结束，删除完成。

再比如删除根节点4，同样先从根节点开始查找，找到了但是不是叶子不能直接删除，知道其右子树的关键字个数大于等于2，所以可以去找右子女去借，将右子树中最小的关键字5删除，再将根节点赋值为5，则间接实现了删除4的目的。删除5同样从根节点开始递归删除，在根节点找，没找到，5比根节点关键字大，故在右子树中寻找，发现其比关键字6小，应该在其左子树中，但是发现关键字6的子女都只有一个关键字，这样可能导致坍塌，所以将关键字6及其子女合并，而后递归删除，这次关键字5在叶子节点中可以直接删除，删除完成。

**4.2算法复杂度分析**

B树的搜索过程是一个在节点内搜索和循其某一条路径向下一层搜索交替进行的过程，因此B树的搜索时间与B树的阶数和B树的高度直接有关。B树的索引查找时间复杂度为O(logm(n))m为子树的最大个数

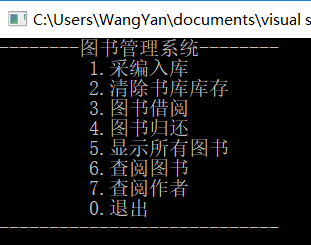
在Myvector中采用下标直接访问时间复杂度为O(1),空间复杂度为O(n),同样添加与删除操作不涉及数据的移动替换，所以时间复杂度同为O(1).

**4.3改进措施：**

系统对用户来说不太友好，因为当查找一本书需要用户输入书的编号，在实际中大多数借阅者不会记录书的号码，更多是名字，应该用映射mapping予以改进，将书号与书名形成一一对应关系。还有录入图书的时候不应该要求输入书的现存量，因为理所应当现存量等于库存量。

五：用户手册：

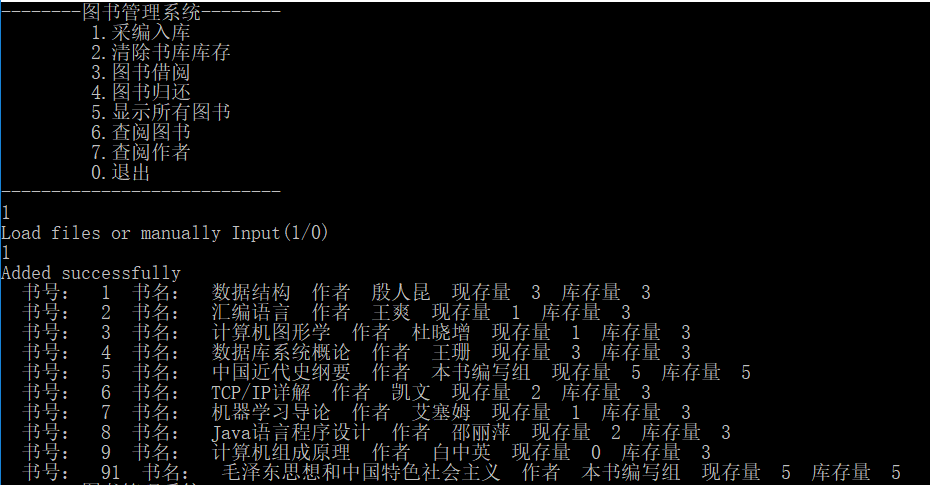
运行程序，屏幕上出现菜单栏，根据不同的需求选择不同的功能。按数字键0退出程序

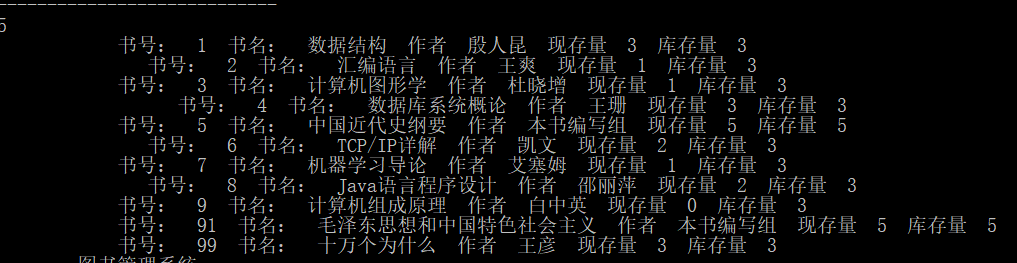
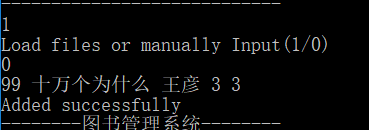


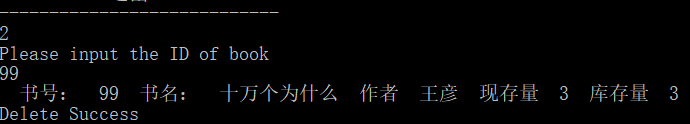
测试数据及测试结果：

5.1检测图书的录入与删除

分别从文件和手动录入图书，然后将全部图书输出，再将一本图书删除，再次打印，查看变化，状态通过。

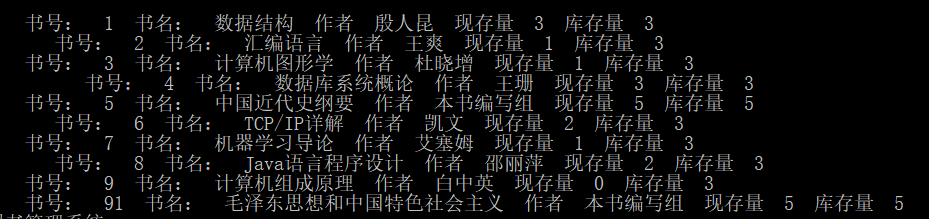


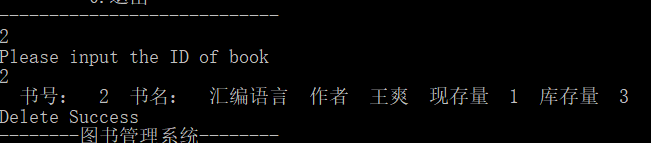


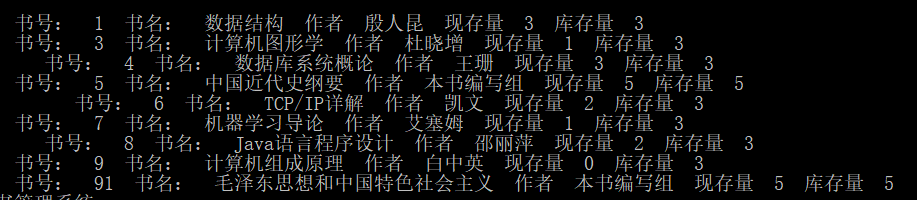


5.2测试B树凹入打印与删除

录入图书，然后将其全部凹入打印，然后再删除某个节点后再次凹入打印，查看删除是否正确(与书上代码删除方式不同)

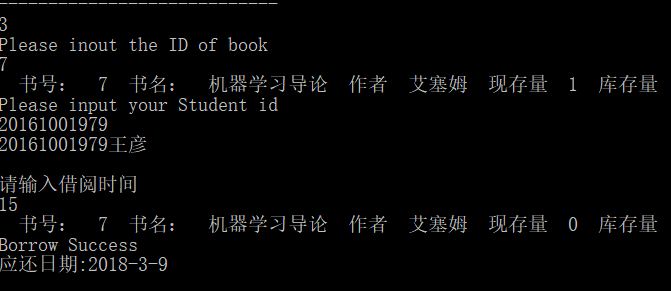


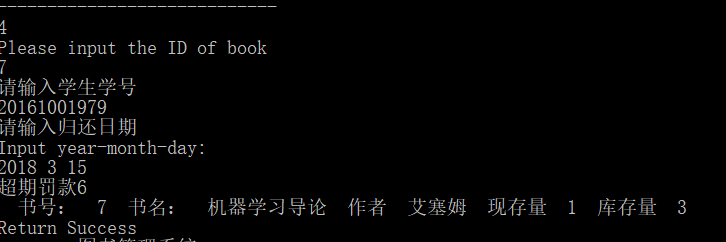




5.3测试图书借阅与归还

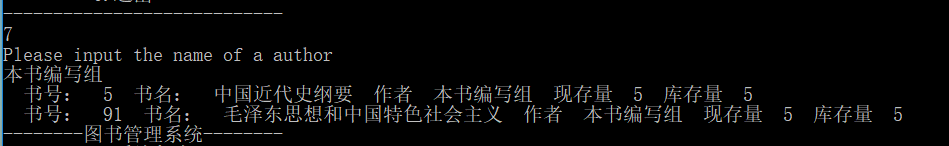
以自己学号借阅一本书，然后归还，查看库存量变化以及超期罚款的处理。状态通过。但是存在不合理的地方，系统的默认时间是2018年2月22日，所有的日期都从这一天开始计算，而且还要学生自己输入借阅日期和归还日期，这就很不合理了，应该获取以下系统的当前时间，而且以更好方式处理借阅日期这个数据，实际中图书馆是规定一个月，这样也比较合理，但是由于是模拟系统，对于还书日期这个数据没有很好的处理方法，只能让用户手动输入了。





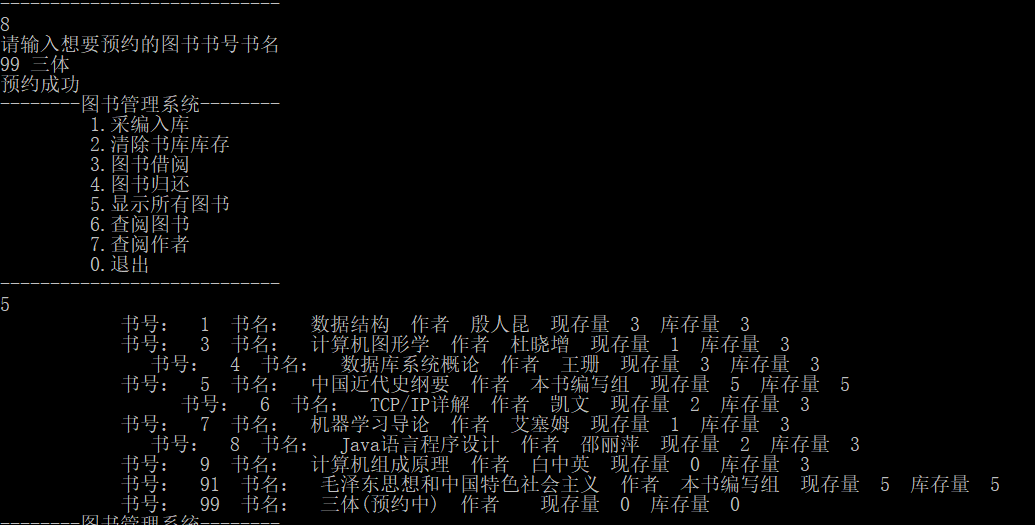
5.4测试作者查询

输入一个作者姓名，查看是否输出其全部著作，状态通过。



5.5预约借书

输入一本图书馆没有的书，将其加入到图书馆库存中(标记为预约),而且库存量现存量为零，打印查看。



七：源程序清单

头文件

Book.h//图书成员函数声明及其数据成员定义

Btree.h//B成员函数声明及其数据成员定义

Date.h//日期成员函数声明及其数据成员定义

Library.h//图书管理系统成员函数声明及其数据成员定义

MyVector.h//容成员函数声明及其数据成员定义

Person.h//人类成员函数声明及其数据成员定义

源文件

Book.cpp//图书成员函数实现

Date.cpp//日期类成员函数实现

Library.cpp//图书馆类成员函数实现

Main.cpp//主函数

Person.cpp//学生类成员函数实现