数据结构课程实习报告

**目 录**

[题目一 电梯模拟 4](#_Toc517338119)

[一、课程设计题目与要求 4](#_Toc517338120)

[1、题目内容 4](#_Toc517338121)

[2、题目要求 4](#_Toc517338122)

[二、需求分析 4](#_Toc517338123)

[1、总体功能需求 4](#_Toc517338124)

[2、功能需求 5](#_Toc517338125)

[3、软件开发平台需求 5](#_Toc517338126)

[4、输入输出要求 5](#_Toc517338127)

[三、设计 5](#_Toc517338128)

[1、设计思想 5](#_Toc517338129)

[2、设计表示： 7](#_Toc517338130)

[3、详细设计 10](#_Toc517338131)

[四、调试分析 17](#_Toc517338132)

[1、遇到主要的问题： 17](#_Toc517338133)

[2、程序的时空复杂度分析： 17](#_Toc517338134)

[3、经验体会： 17](#_Toc517338135)

[五、用户手册 18](#_Toc517338136)

[六、测试数据及测试结果 18](#_Toc517338137)

[七、源程序清单： 20](#_Toc517338138)

[题目二 西文图书管理系统 21](#_Toc517338139)

[一、课程设计题目与要求： 21](#_Toc517338140)

[1、题目内容 21](#_Toc517338141)

[2、题目要求： 21](#_Toc517338142)

[二、需求分析： 21](#_Toc517338143)

[1、总体功能需求 21](#_Toc517338144)

[2、功能需求 22](#_Toc517338145)

[3、软件开发平台需求 22](#_Toc517338146)

[4、输入输出要求 22](#_Toc517338147)

[三、设计 22](#_Toc517338148)

[1、设计思想 22](#_Toc517338149)

[2、设计表示： 24](#_Toc517338150)

[3、详细设计 27](#_Toc517338151)

[四、调试分析 32](#_Toc517338152)

[1、遇到的主要问题： 32](#_Toc517338153)

[2、程序的时空复杂度分析： 32](#_Toc517338154)

[五、用户手册 32](#_Toc517338155)

[六、测试数据及测试结果 33](#_Toc517338156)

[七、源程序清单： 35](#_Toc517338157)

题目一 电梯模拟

### 一、课程设计题目与要求

#### 1、题目内容

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。

#### 2、题目要求

基本：

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：

实现电梯模拟的可视化界面。

### 二、需求分析

#### 1、总体功能需求

该作品将实现一个模拟电梯，在界面上显示出电梯系统状态的变化过程，即发生的全部人和电梯的动作序列，对于乘客而言，动作包括等待电梯到来、进出电梯以及由于等待时间过长而离开电梯，对于电梯而言，动作包括电梯的加减速、上升与下降、开门与关门以及由于在某一层停留时间过长而驶回1层。其中要求乘客的产生是随机的，电梯有一定的载荷量，当超过载荷量的时候，电梯外乘客无法进入。

#### 2、功能需求

能够模拟出正常电梯的行为以及状态，能够计算并打印出当前时间电梯的行为是必要的

#### 3、软件开发平台需求

开发者开发的软件要求能够正常运行在Windows平台，且具有一定的平台兼容性.。为保证软件的上下兼容性，开发者应选择通用的开发工具的来进行开发，目前的开发软件平台为Visual Studio 2016。

#### 4、输入输出要求

该系统无需对其进行输入，用户只需运行，乘客的产生以及各信息由程序随机产生；

系统的输出为电梯系统状态的变化时间过程以及各乘客的动作序列过程，并打印出当前时间，如：

乘客1想做电梯，来自第9楼，去第2楼 现在时间:483

电梯准备去9楼 电梯经过3楼 现在时间:287

### 三、设计

#### 1、设计思想

##### 1、数据结构设计

主要存储结构为类和链表；

电梯内的乘客和电梯外等待的乘客分别由两个链表存储，使用链表存储能够更容易地对乘客进行出入操作，并且可以帮助，当电梯在未超载的前提下，到达某楼层，满足可进电梯条件之后，将该乘客从等待链表中删去，又加入到电梯内乘客的链表中。

乘客和电梯封装成类，其中乘客类的属性包括了乘客的基本信息，包括等待时间、所来和所去楼层以及唯一编号；

电梯类的属性包括电梯从开始运行的时间、存储电梯内外的乘客的两个列表、电梯当前状态以及处于当前状态的时间、上次和当前以及目的楼层、现在电梯运行方向等。其方法包括自定义构造函数、随机产生乘客、主时间函数、乘客进出电梯函数、乘客离开电梯函数、用户因为等待时间过长离开、电梯判断可关门函数、电梯返回一楼函数等；

值得注意的是这里无论是电梯的状态还是乘客的状态都是枚举类型，好处是方便直观。

##### 2、算法设计

该系统可分为产生乘客、乘客进出电梯模块、电梯关门及运动、回到一楼、目的楼层计算五个模块，模拟时钟的运行依次照着这五个模块的次序而运转，接下来会详细介绍这五个模块:

1. 产生乘客模块：

本模块使用随机函数产生乘客，经过测试可得到，平均每100t产生一名乘客较合理，因此，我们对于每1t产生一个随机数，并对结果模上100，若为0则产生乘客，反之不成立。当确定产生乘客时，随机产生乘客的其他信息，譬如乘客当前所在楼层和目的楼层，对其状态设为等待电梯状态并开始计时。

1. 乘客进出电梯模块：

调用此模块之前，需要对当前是否能够开门进行判断，判断可行之后，延时20t来表示开门的20t，此后，再对人进入电梯以及离开电梯进行操作，并将当前楼层修改为之前的目的楼层。

1. 电梯的关门模块：

调用此模块之前，需要对当前是否能够关门进行判断，与上一模块不同的是确定关门之后，每40t判断一次，而不是每时每刻都在判断，关门之后，将当前状态设为waiting或上行、下行，而当前状态时间设为0。

1. 回到一楼模块

该模块相对而言较简单，如果电梯在某层静止时间超过 300t，则驶回 1 层候命，判断可行时，将目的楼层设为1。

1. 目的楼层计算与到达模块

该模块首先会根据当前电梯运转方向、电梯外乘客所在楼层、电梯内乘客的目的楼层来判断是否继续该方向的运转，如果满足，选择最靠近当前楼层的楼层为目的楼层，若不满足，比如，之前电梯方向为上，但是电梯没有想去更高楼层的乘客，电梯外也没有更高楼层的乘客在等侯，这时候，电梯应该考虑向下运转，之后，我们再根据计算结果到达目的楼层。

下图为主程序的流程图：

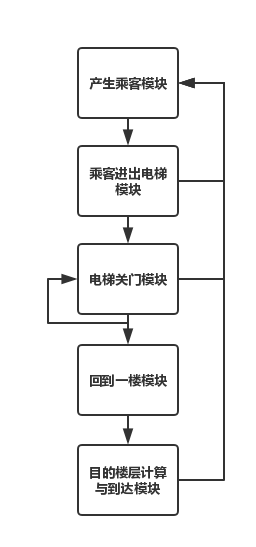


图1 主程序的流程图

#### 2、设计表示：

##### 1、函数调用关系图

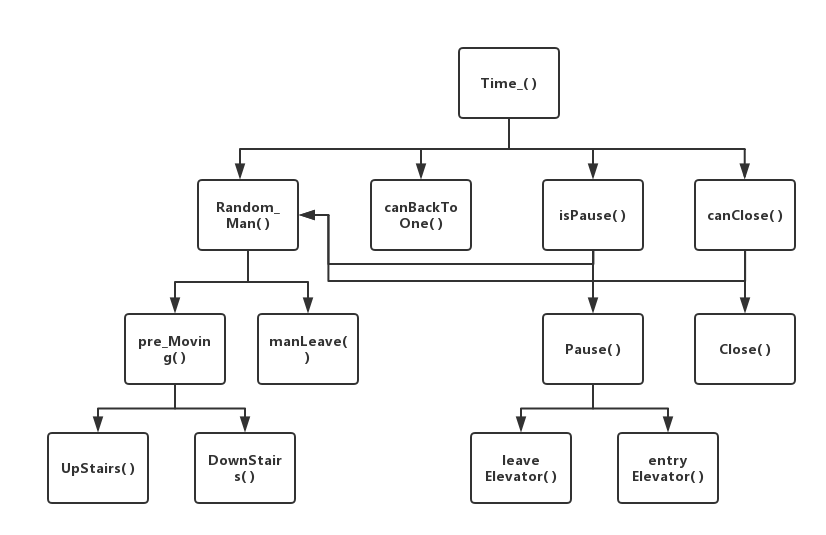


图2 函数调用关系图

##### 2、函数接口规格说明

Elevator()

说 明：重写电梯类的构造函数，初始化电梯的所有参数。

参 数：无

返 回 值：无

void Random\_Man();

说 明：平均每100t随机产生一名乘客，并在该函数中实现乘客的信息随机产生，除此外，还有判断电梯外乘客是否离开，以及通过乘客决定目的楼层。

参 数：无

返 回 值：无

void Time\_();

说 明：电梯的模拟时钟函数，作为主函数中循环调用的函数，并在其内部一次调用其它模块。

参 数：无

返 回 值：无

void entryElevator();

说 明：乘客进入电梯函数 : 在电梯开门后，将满足进入电梯的乘客信息从等待链表中删除，并添加到电梯内部乘客链表中。

参 数：无

返 回 值：无

void leaveElevator();

说 明：乘客离开电梯函数 : 在电梯开门后判断电梯内的乘客是否满足离开电梯的条件，将其信息删去。

参 数：无

返 回 值：无

int canBackToOne();

说 明：判断电梯是否满足因为等待时间过长而回到一楼的条件。

参 数：无

返 回 值：

|  |  |
| --- | --- |
| **值** | **意义** |
| 0 | 不满足自动返回一楼的条件 |
| 1 | 满足自动返回一楼的条件 |

void manLeave();

说 明：对电梯外的乘客依次进行判断是否由于等待时间过长而停止等待

参 数：无

返 回 值：无

int isPause();

说 明：通过计算来判断当前电梯是否已到达目的楼层。

参 数：无

返 回 值：

|  |  |
| --- | --- |
| **值** | **意义** |
| 0 | 电梯未到目的楼层 |
| 1 | 电梯已到目的楼层 |

void Pause();

说 明：当电梯到达到目的楼层之后进行的一系列操作，包括开门、乘客的进入与离开。

参 数：无

返 回 值：无

int pre\_Moving();

说 明：计算电梯当前的目的楼层。

参 数：无

返 回 值：返回当前电梯的目的楼层。

int canClose();

说 明：判断电梯当前是否可以进行关门操作。

参 数：无

返 回 值：

|  |  |
| --- | --- |
| **值** | **意义** |
| 0 | 电梯不可关门 |
| 1 | 电梯可关门 |

void Close();

说 明：当电梯准备关门之后进行的一系列操作，包括关门，以及电梯的方向决定。

参 数：无

返 回 值：无

int UpStairs();

说 明：计算得到电梯内乘客所想去的最高楼层。

参 数：无

返 回 值：返回电梯内乘客所想去的最高楼层。

int DownStairs();

说 明：计算得到电梯内乘客所想去的最低楼层。

参 数：无

返 回 值：计算得到电梯内乘客所想去的最低楼层。

#### 3、详细设计

##### 1、电梯状态

/\*\*

\* 将电梯的状态设为枚举类型，简单直观

\*/

enum Status {

Opening = 1, Opend, Closing, Closed, Waiting, Down\_Moving,Up\_Moving, Decelerate, Accelerate

};

##### 2、乘客类：

/\*\*

\* 定义一个乘客的所有信息

\*/

class Man {

public:

int num;

int from;

int to; //去哪里

int Time; //等待时间

Status\_Man Status; //只有等待和不等待

};

##### 3、电梯类：

/\*\*

\* 自定义的一个模拟电梯类

\*/

数据成员：

public:

list<Man> Pass; //电梯外等待的乘客

list<Man> now\_Pass;//电梯内的乘客

int time; //电梯已经运行的时间

int now\_Floor; //上次电梯开门的楼层

int next\_Floor; //电梯的目的楼层

Status now\_Status; //电梯当前状态

int before; // 1:之前是向上的 0:之前向下

int parm\_Floor; //当前所处楼层

int now\_Status\_Time; //当前状态已花时间

函数成员：

Elevator();

void Random\_Man();

void Time\_();

void entryElevator();

void leaveElevator();

int canBackToOne();

void manLeave();

int isPause();

void Pause();

int pre\_Moving();

int canClose();

void Close();

int UpStairs();

int DownStairs();

/\*\*

\* 对电梯进行初始化

\*/

Elevator::Elevator() {

电梯当前时间 <- 0;

上一次电梯停留楼层 <- 1;

电梯的目的楼层 <- 1;

当前状态 <- Waiting;

当前楼层 <- 1;

之前运行方向 <- 1;

}

/\*\*

\* 电梯的模拟时钟函数，作为主函数中循环调用的函数，并在其内部一次调用其它模块。

\*/

void Elevator::Time\_() {

当前时间++;

当前状态时间++;

暂停1t; //1t

随机产生乘客;

if (是否已到达目的楼层) {

Pause();

}

if (判断是否关门) {

Close();

}

if (是否满足因为等待时间过长而回到一楼的条件) {

next\_Floor <- 1;

cout << "等待时间超300t,现在回一楼\n";

}

}

/\*\*

\* 平均每100t随机产生一名乘客，并在该函数中实现乘客的信息随机产生

\*/

void Elevator::Random\_Man() {

int N =rand() % 100; //平均每100t生成一个人

//printf("\n随机数为%d\n", N);

if (!N)

{

cout << now\_Status<<"\*\*\*\*\n";

if (now\_Status == Waiting) {

cout << "电梯开始加速\n";

now\_Status\_Time = 0;

}

Man add\_Man;

int floor\_from = 1+(rand() % 9);

int floor\_to = 1 + (rand() % 9);

if (floor\_from == floor\_to)

return;

add\_Man.from = floor\_from;

add\_Man.to = floor\_to;

add\_Man.Time = 0;

add\_Man.Status = Waiting\_peo;

add\_Man.num = count1++; //全局变量，指用户的序号

Pass.push\_front(add\_Man);

判断电梯外乘客是否离开;

计算目的楼层;

}

}

/\*\*

\* 当电梯到达到目的楼层之后进行的一系列操作，包括开门、乘客的进入与离开

\*/

void Elevator::Pause() {

now\_Status\_Time <- 0;

now\_Status <- Opening;

For var 0 to 20 do //电梯开门操作

当前时间++;

当前状态时间++;

暂停1t; //1t

随机产生乘客;

End

leaveElevator(); //人离开电梯

entryElevator(); //人进入电梯

now\_Floor <- next\_Floor;

}

/\*\*

\* 当电梯准备关门之后进行的一系列操作，包括关门，以及电梯的方向判断

\*/

void Elevator::Close() {

now\_Status <- Closing;

For var 0 to 20 do //电梯关门操作

当前时间++;

当前状态时间++;

暂停1t; //1t

随机产生乘客;

End

now\_Status <- Closed;

If (无新目的楼层)

now\_Status = Waiting;

Else If (目的楼层>此时楼层)

{

now\_Status = Up\_Moving;

}

Else //目的楼层<此时楼层

now\_Status = Down\_Moving;

now\_Status\_Time <- 0;

}

/\*\*

\* 判断电梯外乘客是否离去

\*/

void Elevator::manLeave() {

For iter Pass.begin() to end() do //电梯关门操作

If(等待时间过长)

从列表删除该乘客

End

}

/\*\*

\* 计算电梯的下一个目的楼层

\*/

int Elevator::pre\_Moving() {

int nex;

if (之前方向为向上) {

nex = 电梯内乘客所想去大于现在楼层的最低楼层;

nex = nex < next\_Floor ? nex : next\_Floor; //与目的楼层作比较，选取最小值

for (; iter1 != Pass.end(); iter1++) //遍历电梯外乘客

{

if (有乘客在nex与当前楼层间，且想去更高层)

nex <- iter1->from;

}

if (电梯无需向上) {

nex = DownStairs(); //遍历电梯外乘客谁想乘坐电梯

list<Man>::iterator iter = Pass.begin();

for (; iter != Pass.end(); iter++)

{

if (iter->from > iter->to && iter->from >= nex )

nex = iter->from;

}

}

}

else { //之前方向为向下

nex = 电梯内乘客所想去小于现在楼层的最高楼层;

nex = nex > next\_Floor ? nex : next\_Floor; //与目的楼层作比较，选取最大值

list<Man>::iterator iter1 = Pass.begin(); //遍历电梯外乘客

for (; iter1 != Pass.end(); iter1++)

{

if (有乘客在nex与当前楼层间，且想去更低层)

nex = iter1->from;

}

if (电梯无需向下) {

nex = UpStairs(); //遍历电梯外乘客谁想乘坐电梯

list<Man>::iterator iter = Pass.begin();

for (; iter != Pass.end(); iter++)

{

if (iter->from < iter->to && iter->from < nex)

nex = iter->from;

}

}

}

return nex; //返回目的地址

}

##### 3、主程序设计：

void main() {

Elevator zzz\_ele = Elevator();

while (1)

{

zzz\_ele.Time\_();

}

}

### 四、调试分析

#### 1、遇到主要的问题：

1） 在使用STL对链表里面的元素进行删除时，总是造成不可预期的错误，因为erase之后，链表更新，迭代器可能发生错误，之前在这个问题上停留了很久，最终通过阅读《STL源码剖析》找到了解决方案，先判断迭代器是否为最后一个元素，若不是，则使用erase()，若是，则使用pop\_back()。

2) 对该模拟电梯进行设计的时候，因为电梯需要逻辑判断和行为实施同时进行，比如需要在电梯上下运行的同时，不能影响乘客的产生，同时还需要判断是否到达目的楼层，一开始所想的是使用多线程编程，但是C++对多线程很不友好，需要对windows底层进行编程，比较麻烦，因此，最后通过sleep()暂时休眠进程来对每一个t进行实际操作。

3） 在计算下一个目的楼层的时候，通过pre\_Moving()计算之后的结果总是与实际结果不一样，具体表现在当电梯某一楼层时，突然显示到了第14层，通过单步断点，最后发现是计算当前楼层发生了逻辑错误，修改代码之后，将当前楼层加入到类的属性之中，防止接下来的代码编写出现类似问题。

4) 显示电梯和用户的动作序列时，之前是将其直接输出，最后显示的太过混乱，电梯行为和用户的行为序列互相混淆，因为没有使用图形化界面，最后解决办法是将电梯的行为输出到屏幕左边，乘客的行为输出到右边，中间显示电梯的时间，同时在关于给每一个人一个标识的时候，为了达到标识与记录两种功能，最后选择使用全局数字序号，因为它简洁，能够产生的标识多而且相比较而言更容易实现。

#### 2、程序的时空复杂度分析：

该系统的运行时间是无穷的，当开始启动之后，电梯的每时每刻都在运行或者等待操作，因此在讨论复杂度的时候是对每一个时间t所要完成的任务作复杂度分析，对于本程序，使用最多的算法是对链表的操作，主要是插入和删除操作，因为插入操作是直接在链表的头指针直接插入的时空复杂度均为O(1)，因此最高的时间复杂度就是对链表的删除操作，其时间复杂度为O(n)，空间复杂度为O(1)，其中n为链表的长度。对于其他操作，如计算目的楼层时，需要对链表中的元素进行遍历，通过分析，时间复杂度为O(n)，空间复杂度为O(1)，实际情况下，由于每t都有可能随机产生1名乘客，但是乘客的最大等待时间是300t，因此电梯外乘客的链表不会超过300，而电梯内乘客链表由于电梯负载限制最多容纳13人。

#### 3、经验体会：

1） 我认为使用多线程编程能使得该系统更简单: 用虚拟时钟，自己设置一个相对时间，每进行一个动作就将这个动作所用的时间加到一个全局变量中，并且每进行一个动作就将该动作的执行内容输出，通过并行操作之后，实际中的t和编码之后的t是一样的，但是在该系统之中，题目所要求得t和该系统的t是有所不同的，但是C++多线程的处理过程很让人费解，因为对于本道题我更倾向于使用Java，因为对Java的多线程有比较深的理解，可以将本道题没有使用多线程的缺陷运用上去。

2) 理解分析问题的能力得到提高。设计一个应用程序关键是对要求做最准确的把握，也就是说弄清楚需求分析是很重要的。这道题刚开始看，很难理解这道题要达到的目的是什么，如何通过模拟时钟来告知各模块应执行什么操作，同时又要对乘客的进入进行操作，这之间的过程很让人费解，经过仔细的多次读题后，画了个草图，才理清其中的思路。所以，通过这道题，大大提高了我的理解分析问题能力。

### 五、用户手册

1.打开文件

windows环境下双击执行文件 zzz\_eleva.exe。

2.运行：

（1）界面左边显示的是电梯的动作序列，比如当前楼层、目标楼层和当前方向，或者是电梯的开门、关门、加速和减速等等；

（2）界面右边显示的是乘客的动作序列，包括乘客的随机产生成功、乘客因等待时间过长而离去、乘客进入或离开电梯；

（3）界面中间显示的是当前电梯运行的时间，单位是t。

3.退出：

在任意时刻，操作者都可点击右上角的“X”来退出电梯模拟。

### 六、测试数据及测试结果

测试输入：

数据是程序随机生成，包括产生乘客的时间和楼层信息：

|  |  |  |  |
| --- | --- | --- | --- |
| 电梯时间 | 乘客编号 | 来自楼层 | 目的楼层 |
| 4t | 乘客1 | 9 | 2 |
| 155t | 乘客2 | 3 | 7 |
| 183t | 乘客3 | 9 | 2 |
| 199t | 乘客4 | 3 | 5 |
| 267t | 乘客5 | 9 | 7 |
| 452t | 乘客6 | 7 | 2 |
| 504t | 乘客7 | 2 | 5 |
| … | … | … | … |

测试目的：

1）电梯是否能够正常运行；

2）电梯时间是否正常。

3）乘客是否能够在等待时间内乘坐电梯，并在目的楼层离开电梯；

正确输出：

现在时间:4 乘客1想做电梯，来自第9楼，去第2楼

电梯准备去9楼 电梯经过1楼 ↑ 现在时间:19

电梯准备去9楼 电梯经过2楼 ↑ 现在时间:70

电梯准备去9楼 电梯经过3楼 ↑ 现在时间:121

现在时间:155 乘客2想做电梯，来自第3楼，去第7楼

电梯准备去9楼 电梯经过4楼 ↑ 现在时间:172

现在时间:183 乘客3想做电梯，来自第9楼，去第2楼

现在时间:199 乘客4想做电梯，来自第3楼，去第5楼

电梯准备去9楼 电梯经过5楼 ↑ 现在时间:223

现在时间:267 乘客5想做电梯，来自第9楼，去第7楼

电梯准备去9楼 电梯经过6楼 ↑ 现在时间:274

现在时间:300 1号乘客等待时间过长离开

电梯准备去9楼 电梯经过7楼 ↑ 现在时间:325

电梯准备去9楼 电梯经过8楼 ↑ 现在时间:376

电梯准备去9楼 电梯经过9楼 ↑ 现在时间:427

现在时间:441

电梯已到9楼

1电梯在开门 现在时间:441

现在时间:452 乘客6想做电梯，来自第7楼，去第2楼

现在楼层在9 现在时间:461

现在时间:470

电梯已到9楼

1电梯在开门 现在时间:470

现在时间:490 5号乘客从9楼进入电梯，想去7楼

电梯进人ing

现在时间:504 乘客7想做电梯，来自第2楼，去第5楼

现在时间:504 2号乘客等待时间过长离开

现在时间:509 乘客8想做电梯，来自第4楼，去第5楼

现在时间:515 3号乘客从9楼进入电梯，想去2楼

电梯进人ing

现在楼层在9 现在时间:540

电梯准备去7楼 电梯经过8楼 ↓ 现在时间:546

现在时间:585 4号乘客等待时间过长离开

电梯准备去7楼 电梯经过7楼 ↓ 现在时间:607

实际输出：

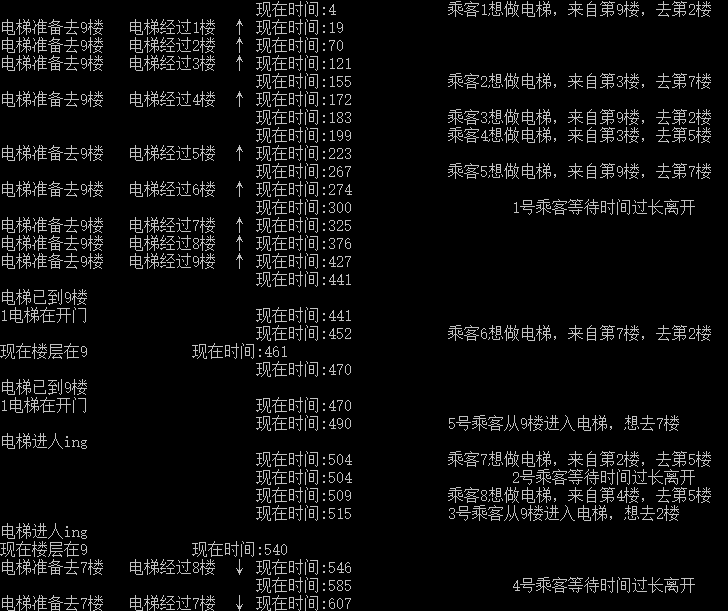


图3 测试结果

当前状态：

通过

### 七、源程序清单：

Elevator.h //包括了电梯类、乘客类的头文件

Ele.cpp //包括了主函数的系统的源文件

Zzz\_eleva.exe //电梯的可执行文件

# 题目二 西文图书管理系统

### 课程设计题目与要求：

#### 1、题目内容

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、题目要求：

（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。要用 B-树（4 阶树）对书号建立索引，以获得高效率。

（3）系统应有以下功能：

采编入库、清除库存、借阅、归还、显示（以凹入表的形式显示）等。

### 需求分析：

#### 1、总体功能需求

该作品旨在实现一个图书管理系统，实现采编入库、查找书籍信息、显示书籍信息、删除书籍信息、借阅图书、归还图书六个功能模块。

在录入书信息模块中，图书馆管理员需要输入满足相应要求的书号、书名、作者以及总量，随后，对其的书号（整形变量）作为索引进行B-树的插入。在查找书籍信息模块，图书馆管理员需要输入书籍的书号，在B-树中寻找，并打印出书籍的信息，包括书名、著者、现存量和总库存量。在删除书籍信息模块，图书管理员可以进行选择：1.删除所有图书，2.删除指定图书，对于第二种选择，程序需要从B-树中删除书籍信息。显示书籍模块要实现以凹入表的形式打印出书籍的书号。

对于借阅者来说，借阅图书和归还图书需要输入相应的图书证号，只有当书存在且可借阅的时候，才能允许借阅，同样地，只有已经借阅，且输入正确的图书证号，才能归还图书。

通过书号构造一个B-树，能够便于图书馆管理员和借阅者查询操作，提高效率。

#### 2、功能需求

能够模拟出图书馆系统的操作，具有良好的人机交互功能，实现采编入库、查找书籍信息、显示书籍信息、删除书籍信息、借阅图书、归还图书六个功能。

#### 3、软件开发平台需求

开发者开发的软件要求能够正常运行在Windows平台，且具有一定的平台兼容性.。为保证软件的上下兼容性，开发者应选择通用的开发工具的来进行开发，目前的开发软件平台为Visual Studio 2016。

#### 4、输入输出要求

该图书管理系统对用户的输入信息有提示，当用户的输入不满足系统条件时，会提醒用户正确地输入，接下来是详细的输入要求：

|  |  |  |  |
| --- | --- | --- | --- |
| **输入信息** | **输入类型** | **输入要求** | **是否可重复** |
| 书号 | 整型 | 输入范围: 1-65535 | 不可重复 |
| 书名 | 字符串 | 最大长度20，以空格结束 | 可重复 |
| 作者 | 字符串 | 最大长度20，以空格结束 | 可重复 |
| 库存 | 整型 | 输入范围: 1-65535 | 可重复 |
| 图书证号 | 整型 | 输入范围: 1-65535 | 不可重复 |

输出要求：在显示图书信息模块，要求对图书进行凹入规则打印。

### 设计

#### 1、设计思想

##### 1、数据结构设计

对于该图书系统而言，逻辑结构由线性结构和树形结构组成。

书籍信息存储在一个B-树上，B-树有利于关键字的查找，它是一种平衡多叉排序树，其优点在于查询快，增删结点相对于链表或者顺序表效率更高，因此用来存储大量图书信息更加合适。

在该系统中能方便图书馆管理员和借阅者进行查询借阅，其中关键字为图书书号，书籍信息中有一个指向借阅者的指针，通过该指针链接下去，可以得到所有该书的借阅者借书证号码，通过使用链表而不是数组的方式存储借阅者信息，可以动态地调整已借阅量，并且利于删除和添加。

B-树上的每一个结点都有两个数组，分别存储孩子结点的指针和该结点上所记录的书籍信息，采用数组的形式存储，为了节省存储空间。另外每个结点还存储指向双亲结点的指针，在插入或者删除书籍时可以减少时间。

##### 2、算法设计

该西文图书管理系统可分为采编入库、查找书籍信息、显示书籍信息、删除书籍信息、借阅图书、归还图书六个模块，通过主程序来对这六个模块进行调度，接下来会详细介绍这六个模块:

1. 采编入库模块：

本模块首先需要图书馆管理员输入新书的信息，再通过书号在B-树搜索该书籍，如果没有找到，则在返回书籍插入的位置之后，调用插入函数对书籍进行插入，若找到该图书，则无法插入。

1. 查找书籍信息

由于本系统的逻辑结构是基于B-树的，是一种平衡多叉排序树，因此，在查找时，需要通过比较所经过结点的关键字，从根节点出发，沿指针搜索结点和结点内进行顺序查找两个过程交叉进行，若查找成功，则返回指向被查关键字所在在结点的指针和关键字在节点中的位置；若查找不成功，则返回插入位置

1. 显示书籍信息

以凹入表的形式显示B树，使用递归的方法显示每个结点，对于一个结点来说，先按顺序打印每个关键字，再对每个结点的子结点都调用打印函数，值得注意的是，在本系统中，仅打印出结点的书号。

1. 删除书籍信息

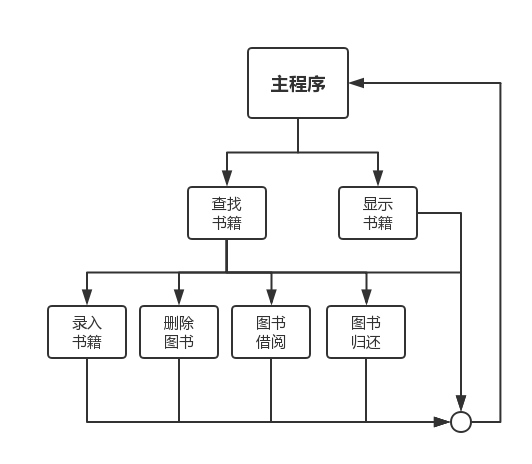
本模块有两类操作，第一类是删除所有书籍信息，在算法中仅需将根节点赋值为NULL，第二类是删除某特定书籍，首先必须找到待删关键字所在结点，并且要求删除之后，结点中关键字的个数不能小于1，否则，要从其左（或右）兄弟结点“借调”关键字，若其左和右兄弟结点均无关键字可借(结点中只有最少量的关键字), 则必须进行结点的合并。

1. 借阅图书模块

对于每一种书籍，若其库存量不为0，那么该书籍可被借阅，借阅证在输入借阅图书后，该模块调用查找模块，判断是否可借，确认可借之后，借阅者输入自己的借阅证号码，若发现该用户已借这类书，则不可再借，相反，算法则将该号码加入到该书的用户链表之中，便完成了图书的借阅。

1. 归还图书模块

进入该模块之后，借阅者将输入书号，若该书的库存等于其总量或不存在该书，则无法归还，反之，输入借阅证号码，再在该书的用户链表中搜索是否该人借阅这本书，若这人的确借阅，则将该借阅者从链表中删去。

下图为主程序的流程图：图4 主程序的流程图

#### 2、设计表示：

##### 1、函数调用关系图

下图为函数调用关系图，其中虚线为类函数：

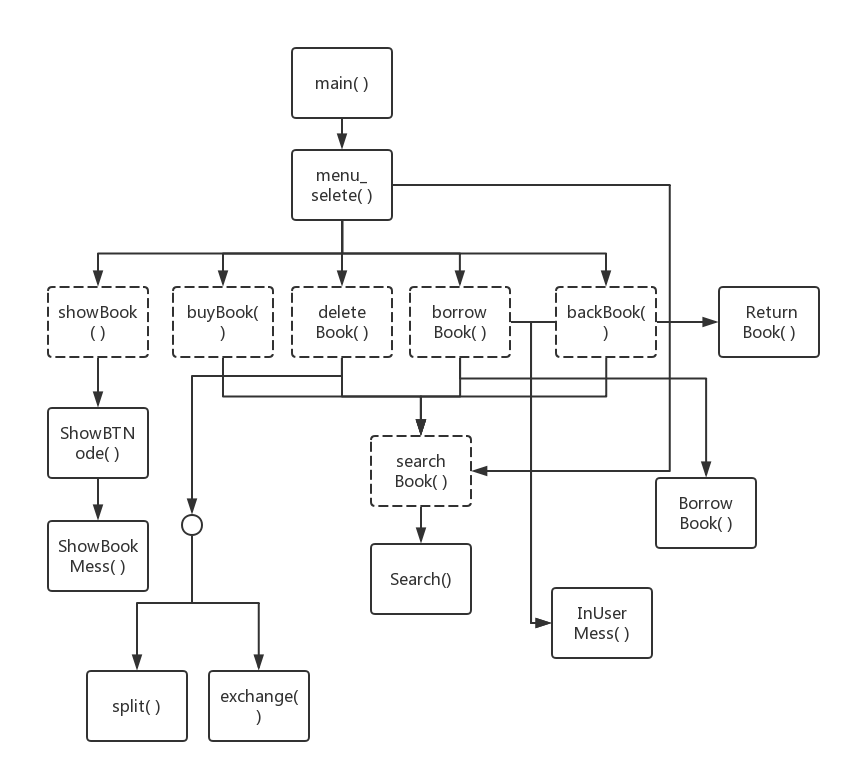


图5 函数调用关系图

##### 2、函数接口规格说明

int ReturnBook(BTree T, Book k)

说 明：注销对借阅者的登记，改变该书的显存量

参 数：存储图书的根节点T，归还图书k

返 回 值：若归还成功则为1，反之为0

int combine(BTree &root, BTree &p);

说 明：将两个结点合并成一个结点

参 数：结点p与其兄弟合并，root为p的父结点

返 回 值：无

int fix(BTree &root, BTree p);

说 明：调整树的结构，判断能不能借，可以借就直接返回

参 数：p为待向左右结点借的结点，root为父结点

返 回 值：无

void InBookMess(Book &book)

说 明：输入书的具体信息

参 数：待输入书book

返 回 值：无

void InBookKey(Book &book)

说 明：输入书的书号

参 数：待输入书book

返 回 值：无

void ShowBookMess(Book book)

说 明：显示书的具体信息,如果书存在就显示

参 数：待显示具体信息的书籍book

返 回 值：无

string ShowBTNode(BTree p);

说 明：显示单个结点中所包含的全部信息

参 数：待显示信息的结点p

返 回 值：含具体信息的字符串

int Search(BTree p, Book K);

说 明：查找在某个结点中的位置

参 数：结点p，待查找书籍K

返 回 值：书籍在结点的位置

Result SearchBTree(BTree T, Book K);

说 明：在m阶B树T上查找关键字K，返回结果(pt,i,tag)

参 数：树根T，关键字的书K，

返 回 值：若查找成功，则特征值 tag=1, 指针 pt所指结点中，第i个关键字等于K;否则特征值 tag=0，等于K的关键字应插入在指针pt所指结点中第i个和第i+1个关键字之间

void Insert(BTree &q, int i, Book x, BTree ap);

说 明：在结点的指定位置插入一条记录

参 数：被插入结点q，插入书籍x，插入位置i,需要加入的子结点ap

返 回 值：无

void split(BTree &q, int s, BTree &ap)

说 明：分裂结点

参 数：被分裂结点q，从位置s之后分裂，分裂后产生结点ap

返 回 值：无

void exchange(BTree &T,int i);

说 明：通过遍历结点，寻找到指定结点的右最左结点，并交换

参 数：待遍历结点T，位置为i

返 回 值：无

int BorrowBook(BTree T,Book k);

说 明：用户借阅的登记与还

参 数：被借阅书籍k，及所在 结点T

返 回 值：借阅成功为1，反之为0

int ReturnBook(BTree T, Book k);

说 明：用户借阅的归还

参 数：被借阅书籍k，及所在 结点T

返 回 值：归还成功为1，反之为0

void BookTranverse(Book &bak,Book k)

说 明：复制结点,将某个结点的值复制到另外一个值上

参 数：被复制结点k，覆盖结点bak

返 回 值：无

#### 3、详细设计

##### 1、借阅者信息

typedef struct User{

unsigned int number; //借书证号码

struct User \*next; //下一个借阅者

}User;

##### 2、图书类

typedef struct Book{

unsigned int key; //图书的书号

char bname[20]; // 书名

char writter[20]; // 著者

unsigned int left; // 现存量

unsigned int total; // 总存量

User \*user; //借阅该书的人

}Book;

##### 3、B- 树结构

typedef struct BTNode{

int keynum; //结点中关键字个数，即结点的大小

struct BTNode \*parent; //指向双亲结点

Book key[Jie+1]; //关键字向量， 0 号单元没有用

struct BTNode \*ptr[Jie+1]; //子树指针向量

}BTNode,\*BTree;

##### 4、图书查找结果结构体

typedef struct{

BTNode \*pt; //指向找到的结点

int i; //1……Jie，在结点中的关键字序号

int tag; //B- 树的查找结果类型

}Result;

##### 图书馆类

数据成员：

public:

BTNode \*root;

函数成员：

bool buyBook(Book my\_book); //图书入库

Result searchBook(Book my\_book); //图书查找

int deleteBook(Book my\_book); //图书删除

int borrowBook(Book my\_book); //图书借出

int backBook(Book my\_book); //图书归还

void showBook(int k); //图书信息打印

/\*\*

\* 图书结点删除

\*/

int Library::deleteBook(Book my\_book) {

if (root == NULL) { //如果还没有录入过书,就直接返回

printf("\t\t\t书库为空!\n");

return 0;

}

BTree p; //查找到的指针

Result R; //查找的结果

int i = 0, sk = 0;

R = searchBook(my\_book);

//如果该 B- 树中不包含该 k 就返回 false

if (R.tag == 0) { //如果查找失败

printf("\t\t\t删除失败!不存在你想要删除的信息!\n");

return 0;

}

p = R.pt; //将查找到的结点赋值给 p

i = R.i; //在结点中的位置

exchange(p, i); //将该结点的值和最右下左角的值交换

if (p->keynum >= (Jie + 1) / 2 - 1) {

return 1;

}

//下面的表示 p->keynum == 0

if (p == root) { //当只有根结点的时候

free(root); //把根结点释放掉

root = NULL; //把删除的结点赋值为空

return 1;

}

while (p) { //其他的情况

if (p == root || p->keynum >= (Jie + 1) / 2 - 1 || !p->parent) //如果p结点删除后还有元素

return 1; //用于循环结束的条件

else { //否则就要向左右两边借元素

if (fix(p->parent, p) == 1) //判断能不能借，可以借就直接返回

return 1;

}

combine(p->parent, p);

p = p->parent;

if ((p == root || !p->parent) && p->keynum == 0) { //如果合并后父结点变成空

root = p->ptr[0];

free(p);

p = root;

root->parent = NULL;

return 1;

}

}

return 1;

}

/\*\*

\* 结点添加

\*/

//返回 false 表示在原有结点上增加数量，返回 true 表示创建了一个新的结点

int InsertBTree(BTree &T, Book K) {

// 在m阶B树T上结点\*q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m阶B树。

BTree ap;

Result R;

BTree q;

int i;

char addnum;

int finished, needNewRoot, s;

// T是空树(参数q初值为NULL)

Book x;

//如果 T 结点为空就生成一个新的结点

if (!T){

NewRoot(T, NULL, K, NULL);

}

else {

//查找元素 k 在树中的位置

R = SearchBTree(T,K);

q = R.pt; //查找到包含元素 k 的结点

i = R.i; //元素 k 在树中的位置

if(R.tag == 1){ //判断该元素在树中是否存在

if(strcmp(q->key[i].bname,K.bname) != 0){

printf("\n\t\t\t录入失败，原因:\n");

printf(".\t\t\t书号冲突，请重新为该书编号!\n\n");

printf("\t\t\t已经存在书号为 %d 的书为:\n",q->key[i].key);

ShowBookMess(q->key[i]);

return 0;

}

printf("\n\t\t\t该书已经存在!\n\n");

ShowBookMess(q->key[i]);

return 0;

}

x = K;

ap = NULL;

finished = needNewRoot = 0;

while (!needNewRoot && !finished) {

Insert(q, i, x, ap); //插入结点

int kk = Search(q->parent, q->key[1]);

cout << "a!!!!!" <<kk<< "\n";

if (q->keynum < Jie)

finished = 1; // 插入完成

else { // 分裂结点\*q

cout << "zzzqd\n";

s = (Jie+1)/2;

split(q, s, ap);

x = q->key[s];

if (q->parent) { // 在双亲结点\*q中查找x的插入位置

q = q->parent;

i = Search(q, x);

}

else

needNewRoot = 1;

}

}

if (needNewRoot) // 根结点已分裂为结点\*q和\*ap

NewRoot(T, q, x, ap); // 生成新根结点\*T,q和ap为子树指针

}

return 1;

}

### 四、调试分析

#### 1、遇到的主要问题：

1）进行编码时，对图书入库进行操作时，算法将一本书插入之后，发现结点已满不可插入，以为要向兄弟结点移动关键字，花费了很长世间，发生错误的根本原因是将B-树的插入和B-树的删去弄混淆了，B-树的插入是向上分裂，而没有向兄弟结点移动的操作，以后编码时，一定要对算法原理在纸上画一下。

2）对非叶结点进行删除操作的时候，发现在交换之后发现了很大错误，B-树的结构变得很混乱，最后找到原因是，判断能不能借，若可以借没有返回，导致继续进行之后的操作。最后经过调试之后，在判断之后，加了一个return，停止函数继续操作。

3）在进行检测时，本图书系统出现了结点分裂时的错误，就是没有考虑到在分裂结点时，该结点的孩子结点的parent指针的改变，通过调试和改正，测试正确。

#### 2、程序的时空复杂度分析：

本图书系统中，需要频繁地进行查找操作，因为阶数是4，在B- 树中进行查找时，其查找时间主要花费在搜索结点上，即主要取决于B- 树的深度。

在含N个关键字的B- 树上进行一次查找，需访问的结点个数不超过log ((N+1)/2)+1，以2为底数。这个公式保证了B-树的查找效率是相当高的。相比较于传统的图书馆系统，图书查找缩小的了对数级。

而对于其他操作，如关键字的增加与删除，也是基于图书的搜索，因此插入和删除的读写盘的次数为O(log2n)。

### 五、用户手册

1.打开文件

windows环境下双击执行文件 zzz\_lib.exe。

2.运行：

（1）进入界面，显示菜单；

（2）根据菜单选择选项:

1.录入书信息

依次输入书号、书名、作者和总量，若输入合法，则入库成功；

2.查找书信息

按照提示输入书号，若输入合理，则返回查找书籍信息；

3.显示书信息

以凹入表的形式显示所有图书的书号；

4.删除书信息

有两种操作：

1：删除所有书籍信息

2：删除某特定书籍，按照提示输入

5.借阅图书

借阅证在输入借阅图书，判断是否可借，确认可借之后，借阅者输入自己的借阅证号码，完成了图书的借阅；

6.归还图书

借阅者将输入书号，再输入借阅者号，完成图书归还；

7.退出系统

3.退出：

在任意时刻，操作者都可点击右上角的“X”来退出图书系统，或在运行菜单选择‘7’。

### 六、测试数据及测试结果

测试输入：

1 1201 时间简史 霍金 8

1 1701 西游记 吴承恩 4

1 1801 高等数学 同济大学 7

1 2001 红楼梦 曹雪芹 6

1 1001 再见的意义 乃木坂46 6

1 1452 Aitakata AKB48 48

1 2738 标准日本语 秋元康 3

1 920 信息论基础 邮电出版社 4

1 2739 时间简史 霍金 8

1 200 西游记 吴承恩 4

1 300 高等数学 同济大学 7

1 100 时间简史 霍金 9

1 400 红楼梦 曹雪芹 6

1 500 再见的意义 乃木坂46 6

1 600 Aitakata AKB48 48

1 700 标准日本语 秋元康 3

1 800 信息论基础 邮电出版社 4

1 1800 12月的袋鼠 SKE48 18

1 1700 Lemon\_Juice HKT48 1

1 1600 欲望者 NMB48 16

1 1500 森林公园 NGT48 43

1 1400 暗暗 STU48 08

1 1300 11月的脚链 AKB48 50

1 1200 Synchron 乃木坂46 20

1 1100 二个人的季节 举坂46 2

1 1000 初日 BNK48 4

1 900 幸运曲奇 JKT48 12

删除节点:

1 1400 暗暗 STU48 08

1 400 红楼梦 曹雪芹 6

1 2738 标准日本语 秋元康 3

测试目的:

1) 判断图书管理系统的插入和删除是否合理；

2) 程序是否崩溃或不希望的闪退；

正确输出：

(书号为:500,----书号为:1001,----书号为:1452)

+--->(书号为:200)

+--->(书号为:100)

+--->(书号为:300)

+--->(书号为:800)

+--->(书号为:600,----书号为:700)

+--->(书号为:900,----书号为:920,----书号为:1000)

+--->(书号为:1201)

+--->(书号为:1100,----书号为:1200)

+--->(书号为:1300)

+--->(书号为:1701,----书号为:2001)

+--->(书号为:1500,----书号为:1600,----书号为:1700)

+--->(书号为:1800,----书号为:1801)

+--->(书号为:2738)

实际输出：



图6 图书馆书籍打印

当前状态：

通过

### 七、源程序清单：

案例.txt //测试用例

btree.cpp //B-树函数实现

btree.h //B-树头文件

main.cpp //主函数

user.cpp //借阅者函数实现

user.h //借阅者头文件