

**北京工业大学**

2018-2019学年 第1学期

信息学部软件学院

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称： | 数据结构与算法课设 | | |
| 报告性质： | 实验报告 | | |
| 组长学号： | 17080224 | 组长姓名： | 孟宁宁 |
| 专业： | 软件工程 | 班级： | 170803 |
| 任课教师： | 陈洪丽 | 课程性质： | 学科基础必修课 |
| 学分： | 2 | 学时： | 60 |
| 完成日期： | 2018年 月 日 | 成绩： |  |
| 小组成员： | 田巍然 胡潇文 郭尚 | | |
| 教师评语： |  | | |

目录

[双层停车场管理系统](#_Toc22247_WPSOffice_Level1) [4](#_Toc22247_WPSOffice_Level1)

[一、 需求分析](#_Toc24273_WPSOffice_Level1) [4](#_Toc24273_WPSOffice_Level1)

[二、 总体设计](#_Toc26740_WPSOffice_Level1) [4](#_Toc26740_WPSOffice_Level1)

[三、 详细设计](#_Toc2220_WPSOffice_Level1) [5](#_Toc2220_WPSOffice_Level1)

[四、 程序调试与测试](#_Toc21517_WPSOffice_Level1) [13](#_Toc21517_WPSOffice_Level1)

[五、 测试结果与结果分析](#_Toc893_WPSOffice_Level1) [17](#_Toc893_WPSOffice_Level1)

[六、 用户使用说明](#_Toc16228_WPSOffice_Level1) [18](#_Toc16228_WPSOffice_Level1)

[七、 课设总结](#_Toc28731_WPSOffice_Level1) [18](#_Toc28731_WPSOffice_Level1)

[八、 参考文献](#_Toc9861_WPSOffice_Level1) [18](#_Toc9861_WPSOffice_Level1)

[九、 附录](#_Toc1513_WPSOffice_Level1) [18](#_Toc1513_WPSOffice_Level1)

[全国交通咨询模拟](#_Toc2791_WPSOffice_Level1) [25](#_Toc2791_WPSOffice_Level1)

[1、设计目的与内容(需求分析)](#_Toc21661_WPSOffice_Level1) [25](#_Toc21661_WPSOffice_Level1)

[2、算法的基本思想](#_Toc26332_WPSOffice_Level1) [27](#_Toc26332_WPSOffice_Level1)

[3、详细设计](#_Toc13740_WPSOffice_Level1) [28](#_Toc13740_WPSOffice_Level1)

[4、测试数据及测试结果分析 3](#_Toc32133_WPSOffice_Level1)8

[5、用户手册 3](#_Toc31307_WPSOffice_Level1)9

[6、课设总结(心得体会) 3](#_Toc29560_WPSOffice_Level1)9

[7、参考文献 3](#_Toc13349_WPSOffice_Level1)9

8、附录 40

[B-Trees 的实现及分析](#_Toc15731_WPSOffice_Level1) [68](#_Toc15731_WPSOffice_Level1)

[1、设计目的与内容(需求分析)](#_Toc4022_WPSOffice_Level1) [68](#_Toc4022_WPSOffice_Level1)

[2、算法的基本思想](#_Toc7638_WPSOffice_Level1) [71](#_Toc7638_WPSOffice_Level1)

[3、系统调试出现的问题](#_Toc13932_WPSOffice_Level1) [74](#_Toc13932_WPSOffice_Level1)

[4、测试数据及测试结果分析](#_Toc28344_WPSOffice_Level1) [74](#_Toc28344_WPSOffice_Level1)

[5、用户手册](#_Toc7899_WPSOffice_Level1) [77](#_Toc7899_WPSOffice_Level1)

[6、课设总结(心得体会)](#_Toc9679_WPSOffice_Level1) [77](#_Toc9679_WPSOffice_Level1)

[7、参考文献](#_Toc9096_WPSOffice_Level1) [78](#_Toc9096_WPSOffice_Level1)

[8、附录](#_Toc10382_WPSOffice_Level1) [78](#_Toc10382_WPSOffice_Level1)

# 双层停车场管理系统

## 需求分析

【问题描述】：

有一个两层的停车场，每层有6个车位，当第一层车停满后才允许使用第二层。每辆车的信息包括车牌号，层号，车位号，停车时间共4项。其中停车时间按分钟计算。假设停车场初始状态为第一层已经停有4辆车，其车位号依次为1—4，停车时间依次为20，15，10，5。即先将这四辆车的信息存入文件“car.dat”中( 数组的对应元素也要进行赋值 )

【基本要求】：

（1）停车操作：当一辆车进入停车场时，先输入其车牌号，再为它分配一个层号和一个车位号(分配前先查询车位的使用情况，如果第一层有空则必须停在第一层)，停车时间设为5，最后将新停入的汽车的信息添加文件“car.dat”中，并将在此之前的所有车的停车时间加5。

（2）收费管理(取车)：当有车离开时，输入其车牌号，先按其停车时间计算费用，每5分钟0.2元。(停车费用可设置一个变量进行保存)，同时从文件“car.dat”中删除该车的信息，并将该车对应的车位设置为可使用状态(即二维数组对应元素清零)。按用户的选择来判断是否要输出停车收费的总计。

（3）输出停车场中全部车辆的信息。

（4）退出系统

## 总体设计

1. 设计思想

数据结构是软件工程专业的基础核心学科，是实现程序结构优化以及效率优化的算法课程。数据结构是计算机存储、组织数据的方式，是指相互之间存在一种或多种特定关系的数据元素的集合。一般认为，一个数据结构是由数据元素依据某种逻辑联系组织起来的。对数据元素间逻辑关系的描述称为数据的逻辑结构，主要可分为：线性结构、树状结构和图形结构；任何需要计算机进行管理和处理的数据元素都必须按某种方式储存在计算机中，对数据元素在计算机中的储存方式成为数据的储存结构，其基本形式有两种：顺序储存结构和链式储存结构；此外讨论一个数据结构必须同时讨论在该类数据上执行的运算才有意义。在许多类型的程序的设计中，数据结构的选择是一个基本的设计考虑因素。

在本题目中，我们通过分析题目需求，选择线性表的数据逻辑结构以及顺序储存结构来实现双层停车场管理系统的算法设计，线性表结构是是[数据结构](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84/1450" \t "_blank)的一种，一个线性表是n个具有相同特性的数据元素的有限序列。线性表中的个数n定义为线性表的长度，n=0时称为空表。在非空表中每个数据元素都有一个确定的位置，如用ai表示数据元素，则i称为数据元素ai在线性表中的位序。

线性表的相邻元素之间存在着序偶关系。如用（a1，…，ai-1，ai，ai+1，…，an）表示一个顺序表，则表中ai-1领先于ai，ai领先于ai+1，称ai-1是ai的直接前驱元素，ai+1是ai的直接后继元素。当i=1,2，…，n-1时，ai有且仅有一个直接后继，当i=2，3，…，n时，ai有且仅有一个直接前驱。

采用线性表是通过分析后发现，适合解决该问题不需要用到其他逻辑较为复杂的其他数据结构，运用最简单有效的线性表，就能完成对该问题解决方案的优化。

## 详细设计

### 数据结构设计

第一步：声明一个名称为InforCar的结构体，用于储存一辆进入停车场的车位信息：车牌号（整型）、停车时间（整型）和费用（双精度浮点型）并用ICar将一个以struct InforCar的数据类型名用typedef函数进行替代。

//车辆信息结点

typedef struct InforCar

{

string data; //车牌号

int time; //停车时间

double fee; //费用

}ICar;

第二步：再声明一个名称为Car\_P的类，将停车操作、收费管理、停车场初始化、停车场车辆信息输出、查询是否有空车位、停车时间累加、文件写入和文件读出七个功能函数封装在Car\_P类中，同时声明以ICar私有变量\*\*space用于储存车位信息，声明一个双精度浮点型变量sum用于记录停车场收费信息。

//模拟停车场类声明

class Car\_P

{

public:

Car\_P(); //构造函数

~Car\_P(); //析构函数

void ParkCar(Car\_P &pCar,int &n); //停车操作

void ChargeOut(Car\_P &pCar,int &n); //收费管理

void FirstPark(); //停车场初始化

void ShowAll(); //输出停车场中全部车辆的信息

int SearchFor(int &a,int &b); //查询是否有空车位

void AddTime(); //停车时间累加

void OutFile(); //文件写入

void InFile(int n); //文件读出

private:

ICar \*\*space; //车位

double sum;

};

第三步：此步骤分为八个模块，对于七个功能函数、构造函数和析构函数进行定义和算法设计

第一块：构造函数和析构函数的定义

实现功能：将类中数据和空间初始化和结束程序是对空间进行析构和信息清除

代码实现

//构造函数

Car\_P::Car\_P()

{

space=new ICar\*[2];

for(int i=0;i<2;i++)

{

space[i]=new ICar[6];

for(int j=0;j<6;j++)

{

space[i][j].data="0"; //车牌号初始化为零

space[i][j].time=0; //停车时间初始化为零

}

}

sum=0;

}

//析构函数

Car\_P::~Car\_P()

{

for(int i=0;i<2;i++)

delete []space[i];

delete []space;

}

第二块：停车场初始化

实现功能：将停车场初始状态设为为第一层已经停有4辆车，其车位号依次为1—4，停车时间依次为20，15，10，5分钟

代码实现：

void Car\_P::FirstPark()

{

ofstream out; //输出流对象

out.open("car.dat",ios::trunc); //打开文件

int t=20;

string a[4]={"001","002","003","004"}; //初始情况的四辆车车牌号

for(int i=0;i<4;i++)

{

space[0][i].data=a[i];

space[0][i].time=t;

out<<"第"<<1<<"层、第"<<i+1<<"辆车的车牌号:"<<space[0][i].data<<" "<<"停车时间:"<<space[0][i].time<<"分钟"<<endl; //将数据写入到输出文件中

t=t-5;

}

out.close();

}

第三块：停车场车辆信息输出

实现功能：将操作后或未进行操作的停车场信息显示输出，达到停车场的状态输出

代码实现：

//查看停车场车辆情况

void Car\_P::ShowAll()

{

for(int i=0;i<2;i++)

{

for(int j=0;j<6;j++)

{

if(space[i][j].data!="0") //若车牌号不为零

{

cout<<"第"<<i+1<<"层、第"<<j+1<<"辆车的车牌号:"<<space[i][j].data<<" ";

cout<<"停车时间:"<<space[i][j].time<<"分钟"<<endl<<endl;

}

else

cout<<"第"<<i+1<<"层、第"<<j+1<<"个车位为空"<<endl<<endl;

}

}

}

第四块：停车操作

功能实现：将新进入停车场的车辆信息录入到停车管理系统中，按序找到空的停车位将车辆信息录入到车位当中，同时调用增加停车时间AddTime()函数对已停入车辆的停车时长、车位数量进行更新。

代码实现：

//停车操作

void Car\_P::ParkCar(Car\_P &pCar,int &n)

{

string id;

int a,b,c;

int flag=0;

cout<<"请输入车牌号"<<endl;

cin>>id;

while(id=="0")

{

cout<<"输入非法，请重新输入!"<<endl;

cin>>id;

}

for(int i=0;i<2;i++)

{

for(int j=0;j<6;j++)

{

if(space[i][j].data==id)

{

flag=1;

}

}

}

if(flag==0)

{

c=pCar.SearchFor(a,b); //查询空车位

if(c==0)

cout<<"没有多余车位了!"<<endl;

else

{

pCar.AddTime(); //时间累加

cout<<"请将车停到第"<<a+1<<"层，第"<<b+1<<"号车位"<<endl;

space[a][b].data=id;

space[a][b].time=5;

n++;

pCar.OutFile(); //写入文件

}

}

else

cout<<"该车已经在停车场中了!"<<endl;

}

第五块：收取停车费

实现功能：当停在双层停车场的车辆离开时，将车位信息更新，将原车辆信息清除，同时将离开停车场车辆应缴纳的停车费用记录到总费用sum变量中，对停车场收费情况进行统计。

代码实现：

//取车收费管理

void Car\_P::ChargeOut(Car\_P &pCar,int &n)

{

int flag=0;

string s;

cout<<"请输入车牌号:"<<endl;

cin>>s;

for(int i=0;i<2;i++)

for(int j=0;j<6;j++)

{

if(space[i][j].data==s)

{

space[i][j].fee=space[i][j].time\*0.2;

space[i][j].data="0";

space[i][j].time=0;

flag=1;

n--;

pCar.OutFile();

cout<<"停车收费:"<<space[i][j].fee<<"元"<<endl;

sum+=space[i][j].fee;

cout<<"是否输出当前停车场总收入:y/n"<<endl;

char ch='n';

cin>>ch;

if(ch=='y'||ch=='Y')

{

cout<<"停车场总收入:"<<sum<<"元"<<endl;

}

break;

}

}

if(flag==0)

cout<<"未找到目标车辆！"<<endl;

}

第六块：文件的写入和输出

实现功能：

代码实现：

//文件写入

void Car\_P::OutFile()

{

ofstream out; //输出流对象

out.open("car.dat",ios::trunc); //打开文件

for(int i=0;i<2;i++)

{

for(int j=0;j<6;j++)

{

if(space[i][j].data!="0")

{

out<<"第"<<i+1<<"层、第"<<j+1<<"辆车的车牌号:"<<space[i][j].data<<"停车时间:"<<space[i][j].time<<"分钟"<<endl; //将数据写入到输出文件中

}

}

}

out.close(); //关闭文件

}

//文件输出

void Car\_P::InFile(int n)

{

ifstream in("car.dat"); //输入流对象，并打开文件

char ch[100];

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

{

in.getline(ch, 100, '\n'); //从外设读取一行数据

cout<<ch<<endl<<endl;

}

in.close(); //关闭文件

}

第四步：主函数功能实现

首先，设计系统调用的功能界面，设置选择功能的switch函数，然后调用声明的以Car\_P类为数据类型的变量所包含的类函数达到系统功能的实现。

代码实现：

//主函数

int main()

{

Car\_P pCar;

int x,n=4;

int flag=1;

pCar.FirstPark(); //初始化

do

{

cout << "界面设计（略）";

cout << "请输入您的选择：";//功能输入选项

cin>>x;

switch(x) //功能输入判断

{

case 1:

pCar.ParkCar(pCar,n); //停车操作

break;

case 2:

pCar.ChargeOut(pCar,n); //收费管理

break;

case 3:

pCar.ShowAll(); //输出停车场中所有车辆信息

break;

case 4:

pCar.InFile(n); //文件读出

break;

case 5:

cout<<"程序运行结束!"<<endl;

flag=0;

break;

default:

cout<<"输入非法！"<<endl;

}

system("pause");

system("cls");

}while(flag==1);

return 0;

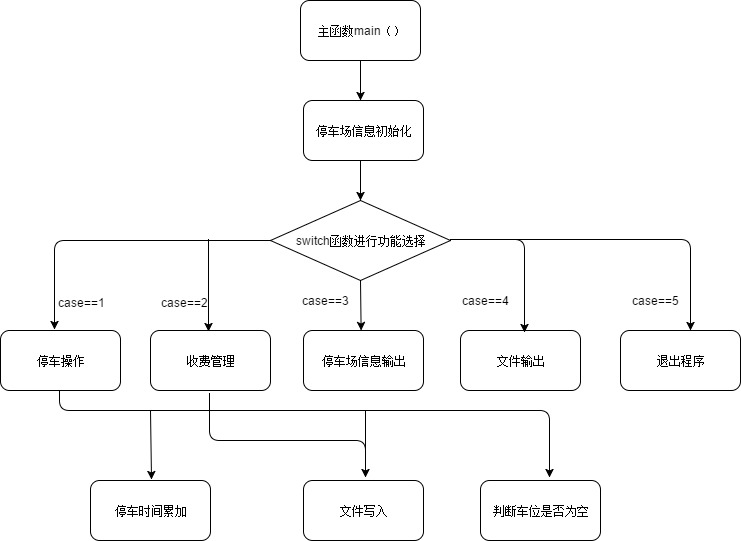
}

第五步：设计系统的退出界面

功能实现：将停车管理系统关闭并且将最终的文件读出。

代码实现：体现在主函数中。

### 2、算法设计

1. 函数调用关系图及说明

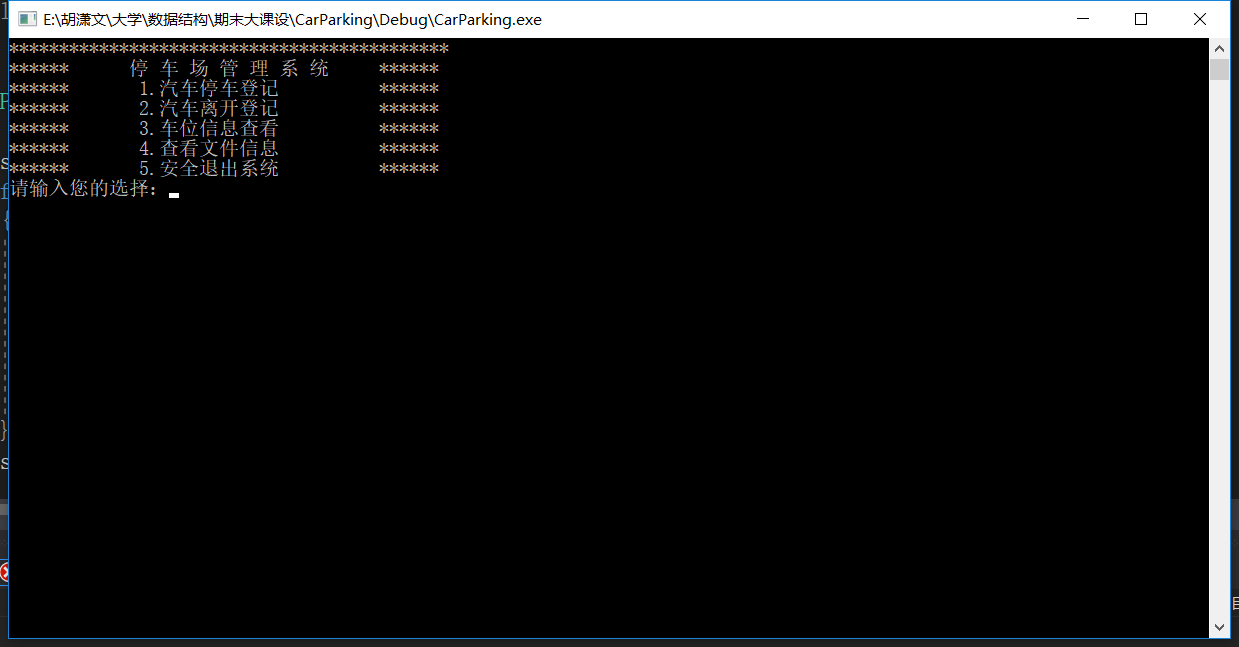
***函数调用图***

1. 函数接口说明

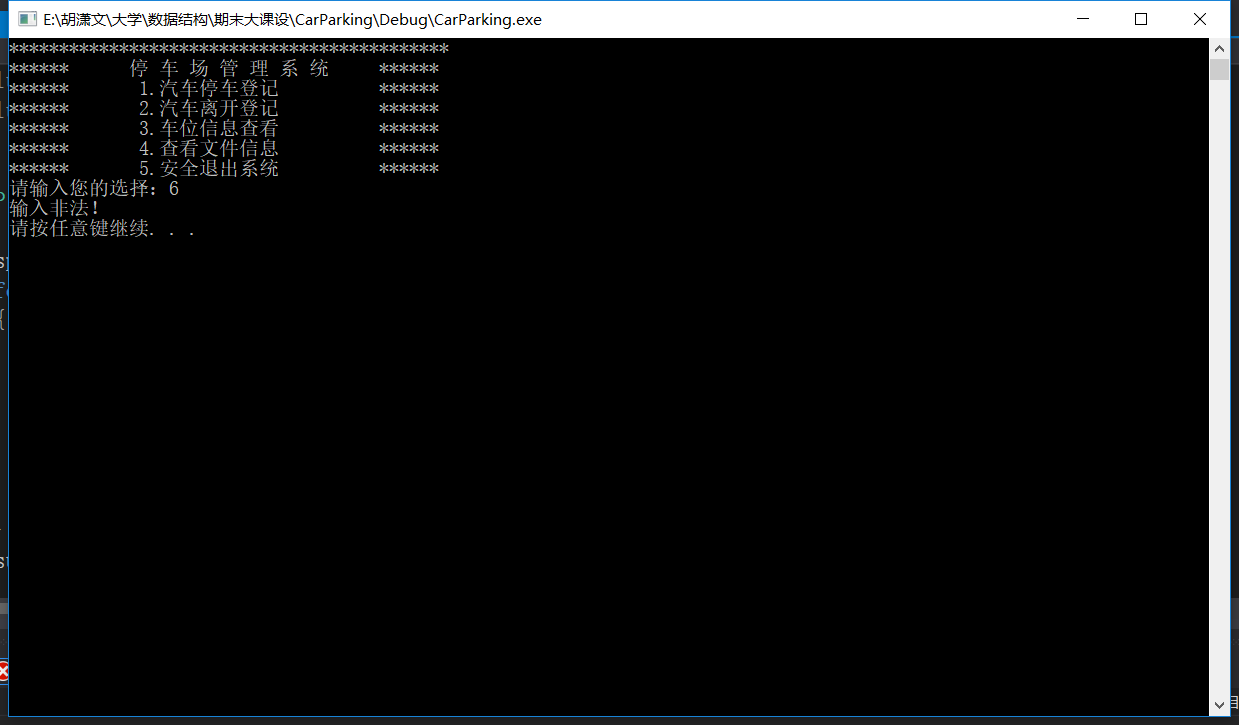
|  |  |
| --- | --- |
| 名称 | 功能 |
| ParkCar() | 停车操作 |
| ChargeOut() | 收费操作 |
| ShowAll() | 停车场信息输出 |
| InFile() | 文件读出 |
| SearchFor() | 判断车位空置状态 |
| AddTime() | 车辆时间累加 |
| OutFile() | 文件写入 |
|  |  |

## 程序调试与测试

注：改调试与测试顺序为按步骤进行的，数据录入为依此的。

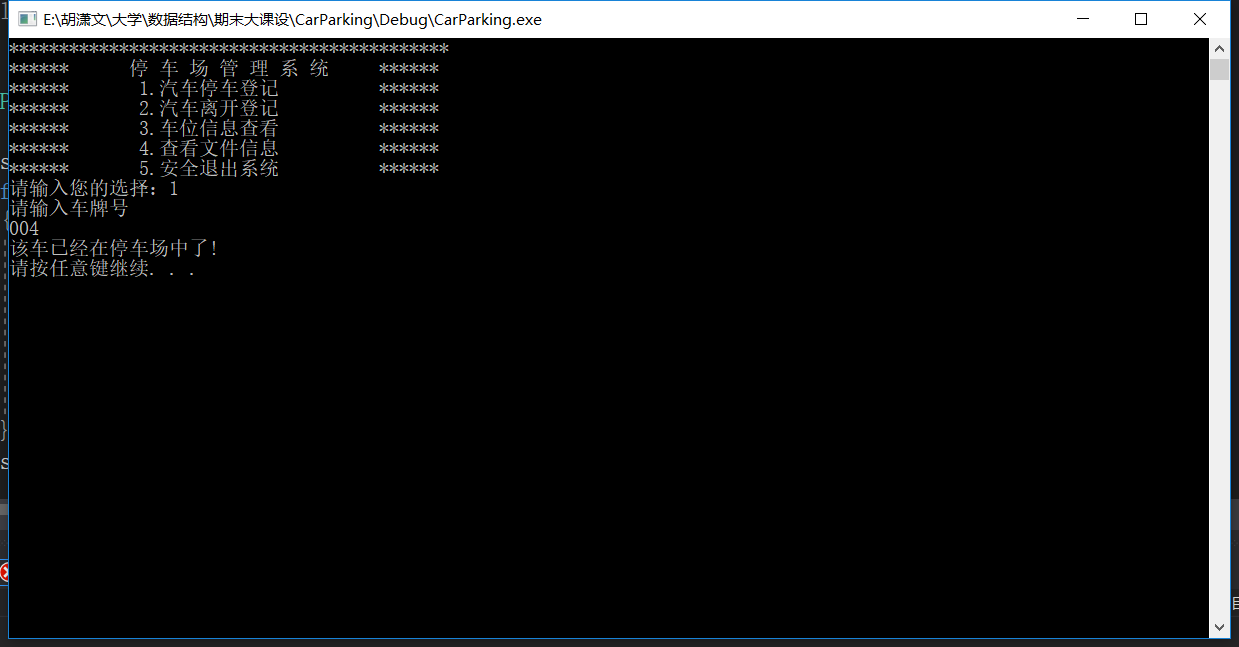


进入界面



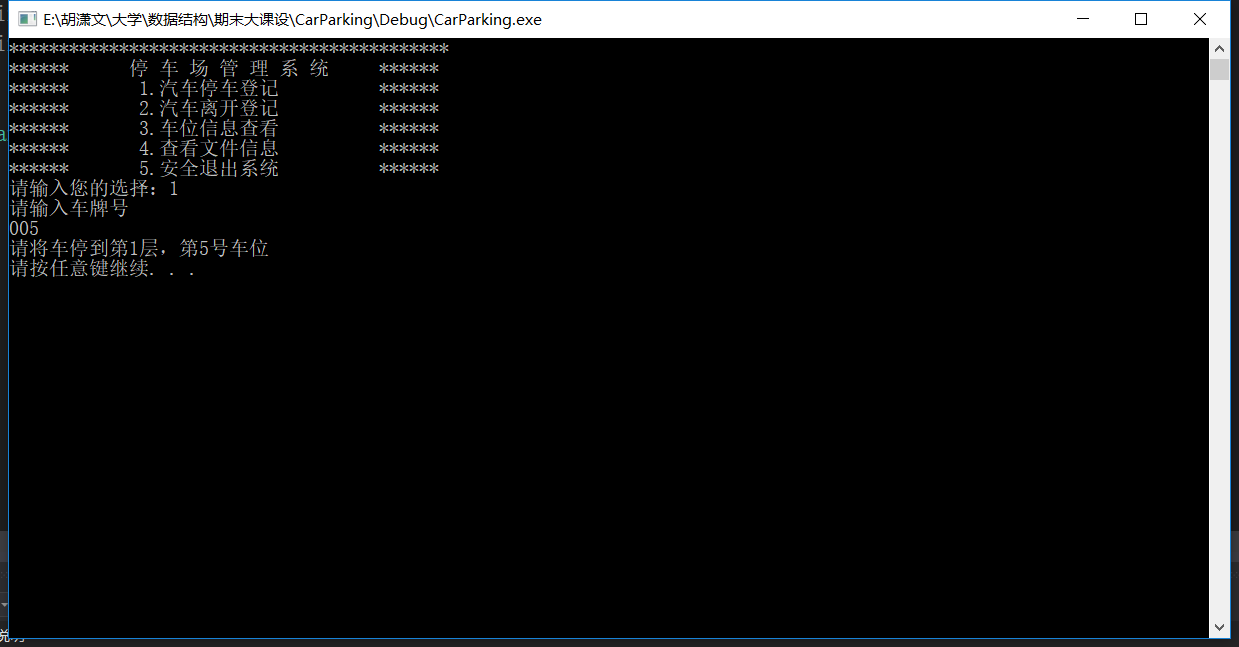
界面报错

输入：6，输出：非法输入



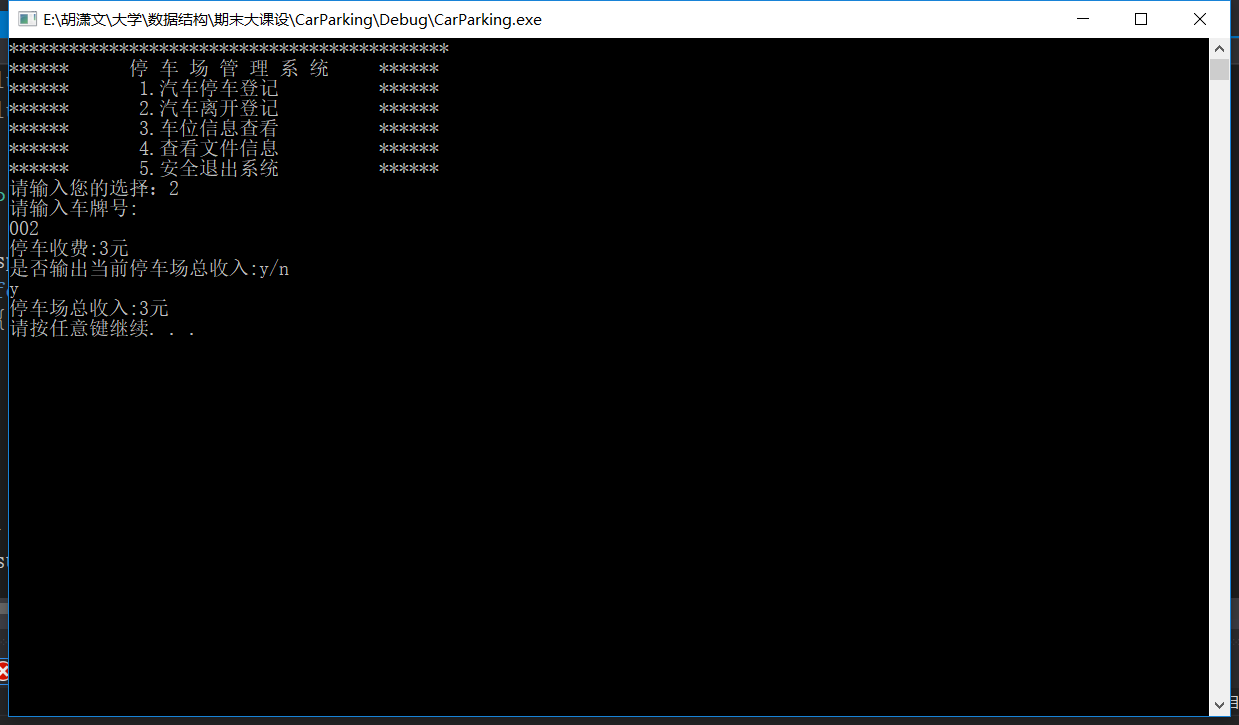
功能一调试（1）

输入：车号004，输出结果：该车已在停车场中



功能一调试（2）

输入：005，输出：请将该车停到第1层，第5号车位

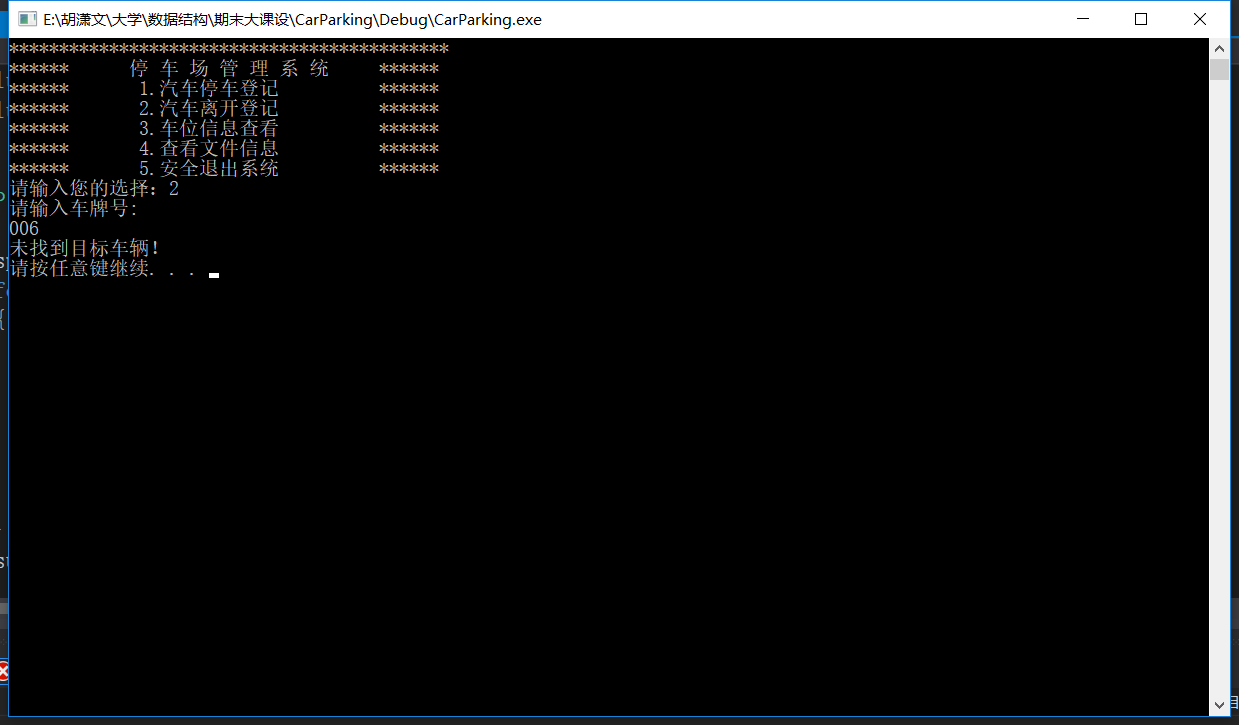


功能二调试（1）

输入车号：002，输出：停车收费3元

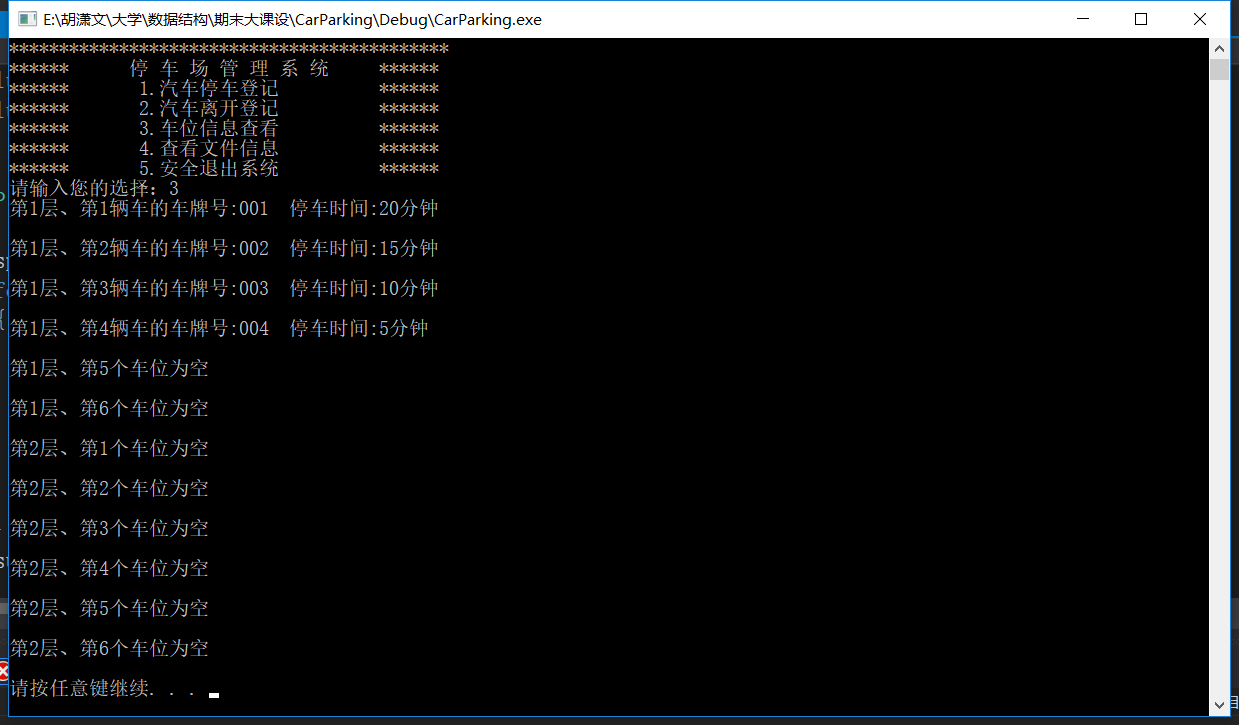
输出：是否输出当前停车场总收入y/n

输入：y，输出：停车场收费3元



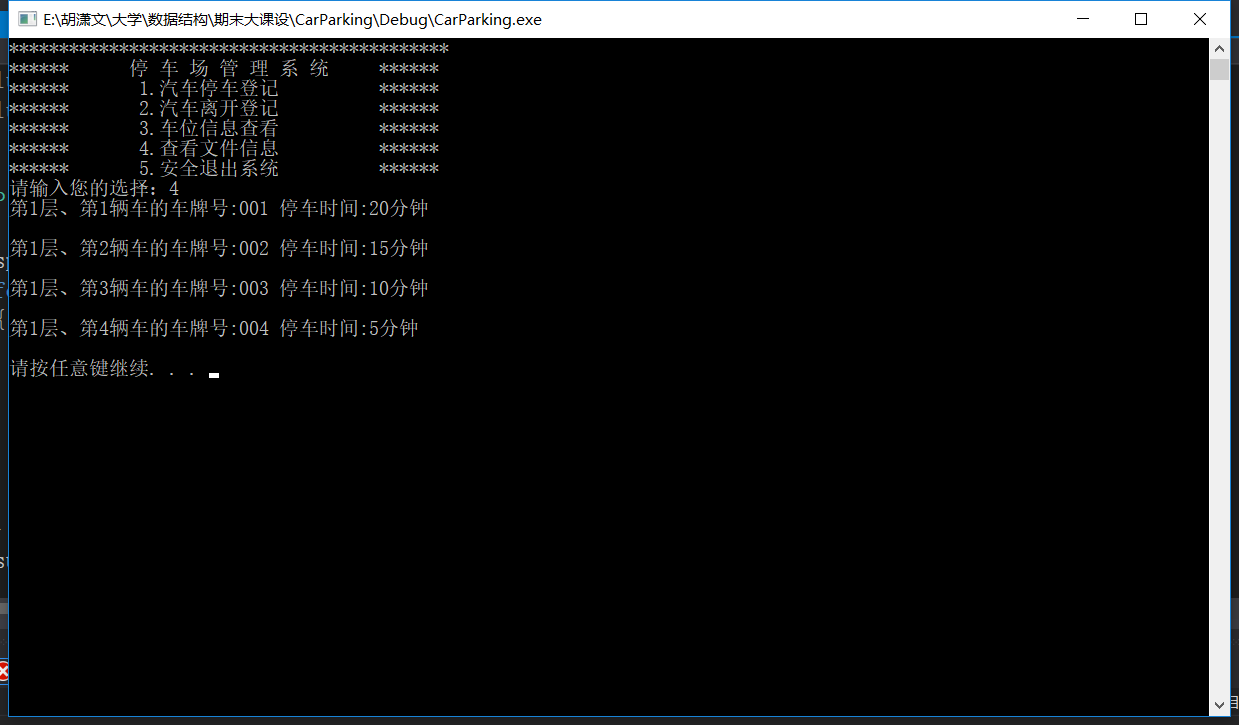
功能二调试（2）

输入车号：006，输出：未找到目标车辆



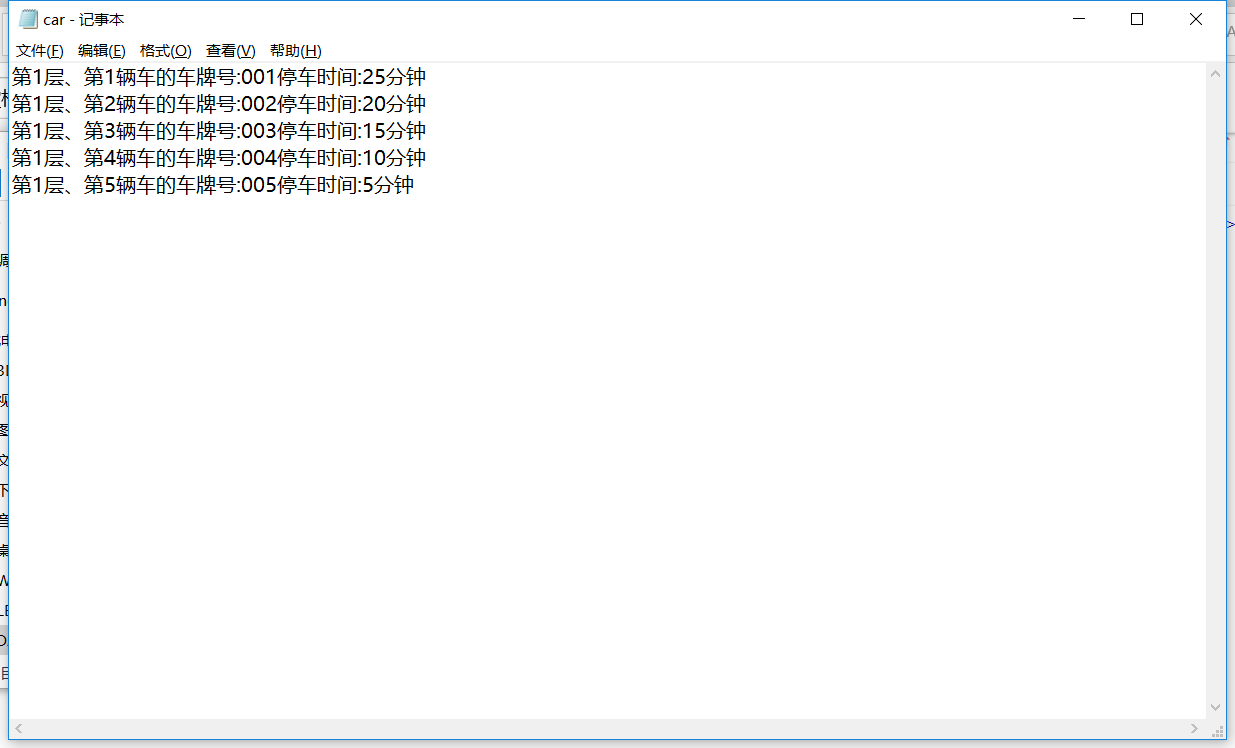
功能三调试

输出结果：将停车场所有车位信息输出显示



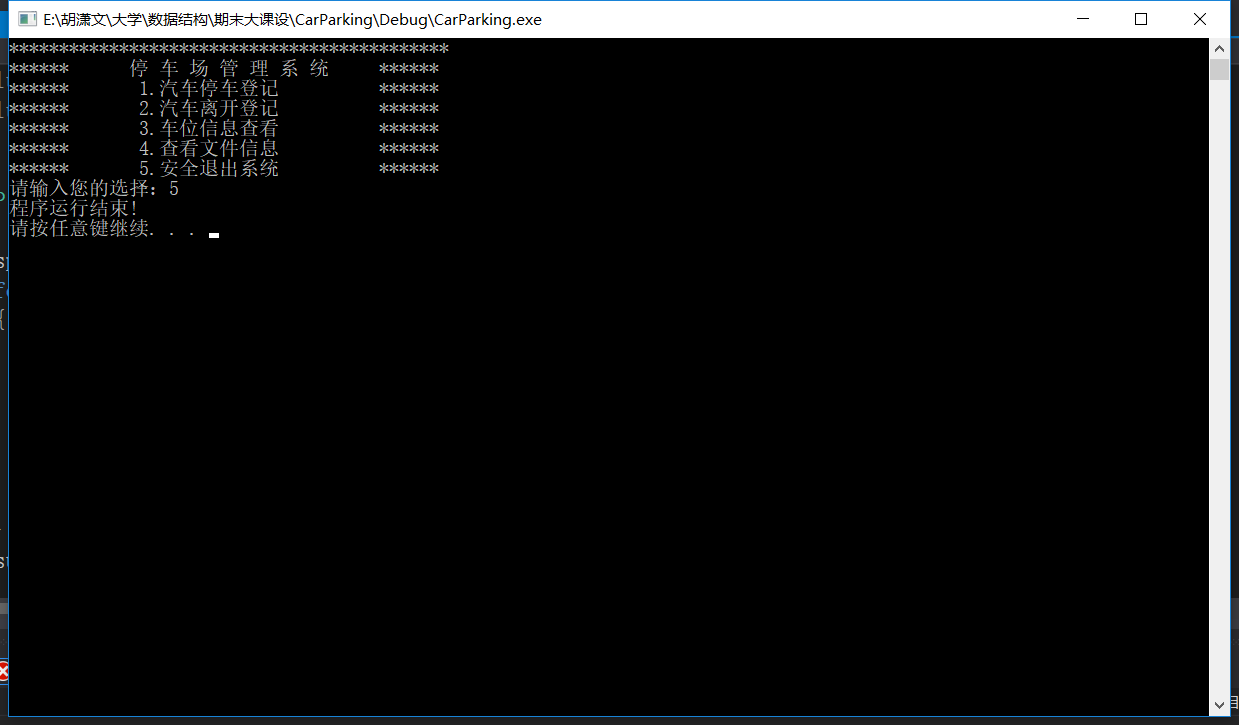
功能四调试（1）

在系统程序中输出文件信息



功能四调试（2）

输出文件的信息，同时查看文件能够直接查看停车信息



功能五调试

直接退出系统，输出“程序运行结束”

## 测试结果与结果分析

调试过程中，发现一些简单的错误，比如引号的不规则使用导致输出结果与设计不符，同时停车操作中对flag==1的判断条件写入有误，导致循环出现错误，输出结果与预期结果完全不一致。同时在最开始停车场初始化的过程中，忘记对文件的初始化，导致最后通过查找无法看到停车场最初始化的信息。

## 用户使用说明

用户通过登陆界面，共有五项功能可供选择：

*第一项“汽车停车登记”*：双层停车场进入新的车辆，对新进入停车场的车辆信息进行录入，对停车场的车位信息进行更新。

*第二项“汽车离开登记”*：当车辆离开时，对离开车辆的信息进行删除，讲停车场的车位信息进行更新，同时对于停车的收费情况进行统计并输出。

*第三项“车位信息查看”*：对于停车场的车位信息和各车辆的停车信息进行输出。

*第四项“查看文件信息”*：会将实时的停车信息以文本的形式输出，在car.dat的文件中查看此时的停车场信息。

*第五项“安全退出系统”*：退出使用双层停车场管理系统，结束本系统的使用。

## 课设总结

胡潇文：对于双层停车场管理的系统设计，最开始的时候我们拿到题目分析，将题目过于复杂化，觉得这道题目无从下手，但是冷静后发现，该题目运用简单的顺序表就能够完成系统功能的基本实现，我们采用二维数组的储存方式进行数据存储，声明类来对函数进行封装，在实现功能的同时，让程序拥有良好的封装性，也让函数的调用变得更加灵活，尤其是私有变量的运用，让程序的安全性大大的增加。在代码编写过程中，我们直接进行了函数的编写，但是在编写过程中发现这样在函数调用方面思路较为混乱，尤其是在ParkCar（）停车操作函数中对于空位查询、时间更新函数的调用的顺序有些混乱。同时在每个函数的报错功能，有些函数中忘记将非法输入设计在函数内，让函数的功能变得不是非常完整，设计逻辑不是非常的严谨。对于本题目最关键的设计就是变量flag的设计，作为判断停车位是否有车停放的判断标识，flag的值为1为0对整个系统程序的影响是关键的，作为了很多函数的和循环的判断依据，基本为flag==1，循环进行。同时，本程序加入了文件写入和输出的功能，实现信息编入文件进行记录的功能，让程序的功能更加完善。

通过编写、调试该项目，我总结出一条以后编程过程中需要非常注意的地方，那就是先画流程图和函数关系调用图，再去进行编写程序，效率高且思路清晰。

## 参考文献

无

## 附录

源代码：

// Car\_Parking.cpp : 定义控制台应用程序的入口点。

#include"stdafx.h"

#include<iostream>

#include<string>

#include<fstream>

using namespace std;

//车辆信息结点

typedef struct InforCar

{

string data; //车牌号

int time; //停车时间

double fee; //费用

}ICar;

//模拟停车场类声明

class Car\_P

{

public:

Car\_P(); //构造函数

~Car\_P(); //析构函数

void ParkCar(Car\_P &pCar,int &n); //停车操作

void ChargeOut(Car\_P &pCar,int &n); //收费管理

void FirstPark(); //停车场初始化

void ShowAll(); //输出停车场中全部车辆的信息

int SearchFor(int &a,int &b); //查询是否有空车位

void AddTime(); //停车时间累加

void OutFile(); //文件写入

void InFile(int n); //文件读出

private:

ICar \*\*space; //车位

double sum;

};

//构造函数

Car\_P::Car\_P()

{

space=new ICar\*[2];

for(int i=0;i<2;i++)

{

space[i]=new ICar[6];

for(int j=0;j<6;j++)

{

space[i][j].data="0"; //车牌号初始化为零

space[i][j].time=0; //停车时间初始化为零

}

}

sum=0;

}

//析构函数

Car\_P::~Car\_P()

{

for(int i=0;i<2;i++)

delete []space[i];

delete []space;

}

//停车场初始化

void Car\_P::FirstPark()

{

ofstream out; //输出流对象

out.open("car.dat",ios::trunc); //打开文件

int t=20;

string a[4]={"001","002","003","004"}; //初始情况的四辆车车牌号

for(int i=0;i<4;i++)

{

space[0][i].data=a[i];

space[0][i].time=t;

out<<"第"<<1<<"层、第"<<i+1<<"辆车的车牌号:"<<space[0][i].data<<" "<<"停车时间:"<<space[0][i].time<<"分钟"<<endl; //将数据写入到输出文件中

t=t-5;

}

out.close();

}

//查看停车场车辆情况

void Car\_P::ShowAll()

{

for(int i=0;i<2;i++)

{

for(int j=0;j<6;j++)

{

if(space[i][j].data!="0") //若车牌号不为零

{

cout<<"第"<<i+1<<"层、第"<<j+1<<"辆车的车牌号:"<<space[i][j].data<<" ";

cout<<"停车时间:"<<space[i][j].time<<"分钟"<<endl<<endl;

}

else

cout<<"第"<<i+1<<"层、第"<<j+1<<"个车位为空"<<endl<<endl;

}

}

}

//增加停车时间

void Car\_P::AddTime()

{

for(int i=0;i<2;i++)

for(int j=0;j<6;j++)

{

if(space[i][j].data!="0")

{

space[i][j].time+=5;

}

}

}

//查询空车位

int Car\_P::SearchFor(int &a,int &b)

{

int flag=0;

for(int i=0;i<2;i++)

{

for(int j=0;j<6;j++)

{

if(space[i][j].data=="0") //从头开始找到的第一个空车位

{

a=i;

b=j;

flag=1;

return 1;

}

}

}

if(flag==0)

return 0;

}

//停车操作

void Car\_P::ParkCar(Car\_P &pCar,int &n)

{

string id;

int a,b,c;

int flag=0;

cout<<"请输入车牌号"<<endl;

cin>>id;

while(id=="0")

{

cout<<"输入非法，请重新输入!"<<endl;

cin>>id;

}

for(int i=0;i<2;i++)

{

for(int j=0;j<6;j++)

{

if(space[i][j].data==id)

{

flag=1;

}

}

}

if(flag==0)

{

c=pCar.SearchFor(a,b); //查询空车位

if(c==0)

cout<<"没有多余车位了!"<<endl;

else

{

pCar.AddTime(); //时间累加

cout<<"请将车停到第"<<a+1<<"层，第"<<b+1<<"号车位"<<endl;

space[a][b].data=id;

space[a][b].time=5;

n++;

pCar.OutFile(); //写入文件

}

}

else

cout<<"该车已经在停车场中了!"<<endl;

}

//取车收费管理

void Car\_P::ChargeOut(Car\_P &pCar,int &n)

{

int flag=0;

string s;

cout<<"请输入车牌号:"<<endl;

cin>>s;

for(int i=0;i<2;i++)

for(int j=0;j<6;j++)

{

if(space[i][j].data==s)

{

space[i][j].fee=space[i][j].time\*0.2;

space[i][j].data="0";

space[i][j].time=0;

flag=1;

n--;

pCar.OutFile();

cout<<"停车收费:"<<space[i][j].fee<<"元"<<endl;

sum+=space[i][j].fee;

cout<<"是否输出当前停车场总收入:y/n"<<endl;

char ch='n';

cin>>ch;

if(ch=='y'||ch=='Y')

{

cout<<"停车场总收入:"<<sum<<"元"<<endl;

}

break;

}

}

if(flag==0)

cout<<"未找到目标车辆！"<<endl;

}

//文件写入

void Car\_P::OutFile()

{

ofstream out; //输出流对象

out.open("car.dat",ios::trunc); //打开文件

for(int i=0;i<2;i++)

{

for(int j=0;j<6;j++)

{

if(space[i][j].data!="0")

{

out<<"第"<<i+1<<"层、第"<<j+1<<"辆车的车牌号:"<<space[i][j].data<<"停车时间:"<<space[i][j].time<<"分钟"<<endl; //将数据写入到输出文件中

}

}

}

out.close(); //关闭文件

}

//文件输出

void Car\_P::InFile(int n)

{

ifstream in("car.dat"); //输入流对象，并打开文件

char ch[100];

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

{

in.getline(ch, 100, '\n'); //从外设读取一行数据

cout<<ch<<endl<<endl;

}

in.close(); //关闭文件

}

//主函数

int main()

{

Car\_P pCar;

int x,n=4;

int flag=1;

pCar.FirstPark(); //初始化

do

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\* 停 车 场 管 理 系 统 \*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\* 1.汽车停车登记 \*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\* 2.汽车离开登记 \*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\* 3.车位信息查看 \*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\* 4.查看文件信息 \*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\* 5.安全退出系统 \*\*\*\*\*\*" << endl;

cout << "请输入您的选择：";

cin>>x;

switch(x)

{

case 1:

pCar.ParkCar(pCar,n); //停车操作

break;

case 2:

pCar.ChargeOut(pCar,n); //收费管理

break;

case 3:

pCar.ShowAll(); //输出停车场中所有车辆信息

break;

case 4:

pCar.InFile(n); //文件读出

break;

case 5:

cout<<"程序运行结束!"<<endl;

flag=0;

break;

default:

cout<<"输入非法！"<<endl;

}

system("pause");

system("cls");

}while(flag==1);

return 0;

}

# 全国交通咨询模拟

【问题描述】：

处于对不同目的的旅客对交通工具有不同的要求。例如，因公出差的旅客希望在旅途中的时间尽可能短，出门旅游的游客则希望旅费尽可能省，而老年旅客则要求中转次数最少。编制一个全国城市间的交通咨询程序，为旅客提供两种或三种最优决策的交通咨询。

【基本要求】：

（1）提供对城市信息进行编辑（如：添加或删除）的功能。

（2）城市之间有两种交通工具：火车和飞机。提供对列车时刻表和飞机航班进行编辑（增设或删除）的功能。

（3）提供两种最优决策：最快到达或最省钱到达。全程只考虑一种交通工具。

（4）旅途中耗费的总时间应该包括中转站的等候时间。

（5）咨询以用户和计算机的对话方式进行。由用户输入起始站、终点站、最优决策原则和交通工具，输出信息：最快需要多长时间才能到达或者最少需要多少旅费才能到达，并详细说明依次于何时乘坐哪一趟列车或哪一次班机到何地。

注：（1）对全国城市交通图和班车时刻表及飞机航班表的编辑，应该提供文件形式输入和键盘输入两种方式。飞机航班表的信息应包括：起始站的出发时间、终点站的到达时间和票价；列车时刻表则需根据交通图给出各个路段的详细信息，例如：对于从北京到上海的火车，需给出北京至天津、天津至徐州及徐州至各段的出发时间、到达时间和票价信息。

（2）以邻接表作交通图的存储结构，表示边的结点内除含有邻接点的信息外，包括交通工具、路程中消耗的时间和花费以及出发和到达的时间等多项属性。

## 1、设计目的与内容(需求分析)

系统分为管理员操作和普通用户操作两方面，管理员可以对城市、交通线进行增删查的操作，用户可以查询费用最省线路和时间最省线路。采用邻接表的方式对线路进行存储，用迪杰斯特拉算法求最省线路。三个文件分别存储城市名（图的顶点），列车线和航班线（图的边）

各模块示意图



## 2、算法的基本思想

进行概要设计和详细设计。说明用到的数据结构定义、主程序的流程及各程序模块之间的调用关系。并用自然语言描述每个模块所涉及的算法。

概要设计（包括选择什么数据结构？数据结构采用哪种存储方式？选择的原因？设计哪些操作？这些操作之间的调用关系等等），即系统总体设计－确定总体框架，确定系统功能模块（或程序功能模块），并给出说明，用文字或框图均可；

由于题目中要求使用邻接表来存储,构建了一个交通工具信息结构TrafficNode和路线信息结构VNode, 交通工具信息结构（即边链表）存储班次、出发时间、到达时间、列车/航班运行时间、到达城市、票价，通过边链指针连接，路线信息结构（即顶点表）存储城市名，以该城市为始发站的列车、航班数量，列车线和航班线均采用交通工具信息结构类型的指针进行连接。并构建一个交通系统类TrafficGraph，其中私有成员包括一个int型变量Cnum用于统计系统中的城市个数，以及一个路线信息结构VNode类型数组AddList，对每个城市的信息按照录入的顺序（数组下标）进行存储。公有成员包括相应的成员函数以实现各项操作。

## 3、详细设计

### (1)设计思想

**1)数据结构设计**

typedef struct TrafficNodeData //交通工具信息

{

char name[MaxSize]; //班次

int StartTime; //出发时间

int StopTime; //到达时间

int Duration; //时间段

int EndCity; //边的另一顶点，即下一个城市编号

int Cost; //票价

struct TrafficNodeData \*link; //下一条边链指针

}TrafficNode;

typedef struct VNodeData //路线信息

{

string cityname ; //城市名称

int TrainNum,FlightNum; //列车、航班数量

TrafficNode \*Tadj; //列车线

TrafficNode \*Fadj; //航班线

}VNode;

class TrafficGraph//交通系统类的定义

{

public:

TrafficGraph(); //构造函数

~TrafficGraph(); //析构函数

int InitialInterface(); //开始及登录界面

void ShowMenu(int n); //功能选择界面

bool ShowFpath(string name);//显示航班线

bool ShowTpath(string name);//显示列车线

bool DelFlight();//删除航班线

bool InsertFlight(int i,int j,int TFcost,int time1,int time2,char \*TFname);//添加航班线

bool DelTrain();//删除火车线

bool InsertTrain(int i,int j,int TFcost,int time1,int time2,char \*TFname);//添加火车线

bool DelCity(string Name);//删除城市

void InsertCity(string Name);//添加城市

int FindCity (string name);//查找城市并返回城市序号

//void InitData();

void FastestRoute(); //耗时最短路径

void CheapestRoute(); //花钱最少路径

int getCost(int v,int u,int choose); //返回两个城市之间的票价

int getDuration(int v1,int v2,int choose); //返回行驶的时间间隔

int getTimes(int v1,int u,int k,int choose); //返回等候的时间

void getInfor(int v1,int v2,int choose); //获取列车或航班信息

int SaveSysInfo(); //将信息存入文件

int InitSysData () ; //用文件信息初始化

private:

VNode \*AddList; //系统信息

int Cnum; // 城市个数

};

**2)算法设计**

**查找费用最省路线：图一：数组集合定义 图二：数组初始化**

**图三：迪杰斯特拉算法计算最小费用 图四：建立栈数组输出各站信息**

图一 图二

**图三 图四**

查找最快路径与之类似，但需要计算等候时间

添加航班线InsertFlight(int i,int j,int TFcost,int time1,int time2,char \*TFname)

## (2)设计表示

用文件信息初始化系统InitSysData( )

**1)函数调用关系图及其说明**

添加火车线InsertTrain(int i,int j,int TFcost,int time1,int time2,char \*TFname)

主函数

int main( )

初始化及登录界面InitialInterface( )

添加城市

InsertCity(name)

输出功能表

ShowMenu(int n)

显示航班线 ShowFpath(name)

删除城市DelCity(name)

显示列车线ShowTpath(name)

将交通系统信息写入文件 SaveSysInfo( )

删除航班线DelFlight( )

删除火车线DelTrain( )

查找城市并返回城市序号FindCity(name)

查找最便宜路线CheapestRoute( )

查找最快路径

FastestRoute( )

获取等候时间 getTimes(int v1,int u,int k,int choose)

获取线路信息 getInfor (int v1,int v2,int choose)

获取两个城市间的费用getCost(int v,int u,int choose)

获取时间段 getDuration(int v1,int v2,int choose)

**2)函数接口说明**

**ShowMenu(int n) 根据选择显示界面**

**添加城市、删除城市、查找城市并返回城市序号、显示线路均传入输入的城市名**

**添加火车、航班线：传入始发站和终点站的下标，票价、出发时间、到达时间和班次**

**获取两个城市间的费用，获取时间段 ：传入始发站和终点站的下标和用户的选择**

**获取等候时间：传入连续三个站的下标和用户的选择**

### (3)详细设计

**初始化及登录界面**

1.文件写入 文件读出

2.主页显示（欢迎内容及用户操作提示）；

3.设置管理员账号及密码

4.输入选择操作：

switch(x)

4.1用户注册:

4.1.1输入用户名

4.1.2输入面膜

for(小于密码位数)

输入的是字符 显示\*；键入回车 密码输入结束

再次输入密码：

while(不一致)

重新输入

用户名、密码存入文件，跳转至4.2

4.2用户登录:

4.2.1输入用户名：

if(读取失败)

退出

do

在文件中查找该用户名是否注册

if(查找成功)

查找标志置1

记录是第几个用户

if(查找标志为0) 循环终止标记为1

else

输入密码

if(正确) 循环终止标记为0

else 循环终止标记为1

while(循环终止标记为1);

4.3管理员登录:

do

输入用户名、密码:

if(正确) 循环终止标记为0;

else 循环终止标记为1;

while(循环终止标记为1);

4.4退出系统:

4.5default:返回该页面

返回最终选择值（传递给功能表界面）;

**添加城市**

传入输入的城市名

1.if(小于系统容量)

1.1AddList数组中录入城市名

1.2列车及航班数量置0;

1.3城市个数+1;

**查找城市并返回城市序号**

传入输入的城市名

1.for(小于城市个数)

1.1if (在AddList数组中找到) 返回下标;

else 返回-1；

**删除城市**

传入输入的城市名

1.if(表空或顶点号超出范围) return false;

2.while(该城市的列车线边链表非空)

2.1取邻接顶点k

2.2找对称存在的边结点s

2.3找s关联的所有结点

2.4if(s!=NULL) 删除对称存在的边结点

2.5清除该城市的边链表结点

2.6边数减一

3.while(该城市的航班线边链表非空) //3.1~3.6同上

4.城市数减一

5.AddList数组填补(将表尾城市填补到删除城市位置）

5.1while(填补城市的列车线边链表非空)

5.1.1指针指向下一站;

5.1.2while(不是该班次的终点站)

if(再下一站是该班次的终点站)

记下该站下标; break;

else 再往后寻找;

5.2while(填补城市的航班线边链表非空) //{同上}

**添加火车线**

**传入始发站和终点站的城市下标，班次名称，出发时间和到达时间，票价**

1.if(始发站和终点站均存在)

1.1指针指向i对应的列车线

1.2while(非空，邻接点不是终点站) 寻找邻接顶点

1.3if(线路已存在)return false;

else创建新结点

1.3.1链入i的列车线

1.3.2插入各项信息

1.3.1线路数++;

**删除火车线**

1.输入要删除的线路的始发站:

2.记录始发站下标;

2.1if(可输出线路)

2.1.1输入你想删除的班次:

2.1.2while(线路表不空)

if(找到对应的线路) 记录终点站线路下标;指针指向下一线路;

2.1.3if(起始站和终点站存在)

while(线路仍存在&&未到达终点站) 指针p指向线路

if(p!=NULL) 指针后移

if(删除的是该线路的第一段) 直接链入之后的线路;

else 与之前的线路相连;

delete p;

2.1.4线路数减一

**航班线的添加与删除同上**

**查找最便宜路线**

1.bool \*S=new bool[n]; //最短路径顶点集

int dist[n]; //dist[i]是当前求到的从顶点v到顶点i的最短路径长度

int path[n]; //path[i]存放求到的最短路径

2.输入起始站和终点站

2.1if(均存在)

2.1.1输入出行方式

2.1.2 for(小于城市个数)

权重数组初始化 顶点集数组初始化

if(当前花费小于预设最大值) path[i]=v1;//记下该顶点下标

else path[i]=-1;

2.1.3 顶点v1加入顶点集合

2.1.4 for(小于除始发站的城市个数)

选不在S中具有最短路径的顶点u

for(小于城市个数)

if(j不在S中且路径长度小于最小值)

记下顶点j;修改最小值;

将顶点u加入集合S

2.1.5 for(小于城市个数)

if（顶点k未加入S，且绕过u可以缩短路径)

修改到k的最短路径

if(路线的花费存在)

int \*d=new int [MAX\_VERTEX\_NUM]; //栈数组

最省路线中的城市顶点倒序入栈

出栈输出

显示总费用及线路信息;

delete []d;

**查找最快路径** 与之类似，因为有等候时间，故设计两个函数计算运行时间和等候时间

**获取时间段**

**传入输入的始发站和终点站下标及选择的出行方式**

1.if(输入的始发站和终点站存在)

1.1if(输入的两个站点相同) return 0；

1.2 选择为1：指针指向火车线；

选择为2：指针指向航班线；

1.3完整便利线路

返回时间段；

**获取等候时间**

**传入输入的始发站v1和修改的结点k下标及线路中间结点u的下标**

1. if（节点存在）

1.1建立指针p，q

1.1.1选择火车：p、q分别指向火车线的v1和u;

1.1.2选择航班：p、q分别指向航班线的v1和u;

1.2while（p未遍历到u） p=p->link

1.3while（q未遍历到k） q=q->link

1.4if(p!=NULL&&q!=NULL) return q->StartTime-p->StopTime

**文件操作**

**1.**在D盘建立三个文本文件

const char CityFile[] ="D:\\city.txt";

const char TrainFile[] ="D:\\train.txt";

const char FlightFile[] ="D:\\flight.txt";

**将交通系统信息写入文件**

1.FILE \*fp;TrafficNode \*p;

2.打开城市文件 读取文件中城市个数Cnum

2.1for (i=0;i<Cnum;i++)

依次读取各城市名

2.2关闭城市文件;total=0;

3.打开列车文件

3.1for (i=0;i<Cnum;i++)

total+=AddList[i].TrainNum;

在文件中打印

3.2for (i=0;i<Cnum;i++)

3.2.1for (列车线不空)

读取各项信息，并在文件中打印出来

3.3关闭列车文件;total=0;

4.打开航班文件

4.1for (i=0;i<Cnum;i++)

total+=AddList[i].FlightNum;

在文件中打印

4.2for (i=0;i<Cnum;i++)

4.2.1for (航班线不空)

读取各项信息，并在文件中打印出来

4.3关闭航班文件;

**用文件信息初始化系统**

1.FILE \*fp;TrafficNode \*p;

char \*a,\*b,\*c;

a=new char;

b=new char;

c=new char;

int i,j,k,hour1,minute1,num,cost,hour2,minute2;

2.打开城市文件

2.1if (!fp) 报错}

2.2for (i=0;i<Cnum;i++)

读取城市名

AddList[i].cityname=p;

2.3关闭城市文件;

3.打开列车文件

3.1if (!fp) 报错

3.2for (i=0;i<num;i++)

读取各项信息

InsertTrain();

3.3关闭列车文件;

4.打开航班文件

4.1if (!fp) 报错

4.2for (i=0;i<num;i++)

读取各项信息

InsertFlight();

4.3关闭列车文件;

## 3、系统调试出现的问题

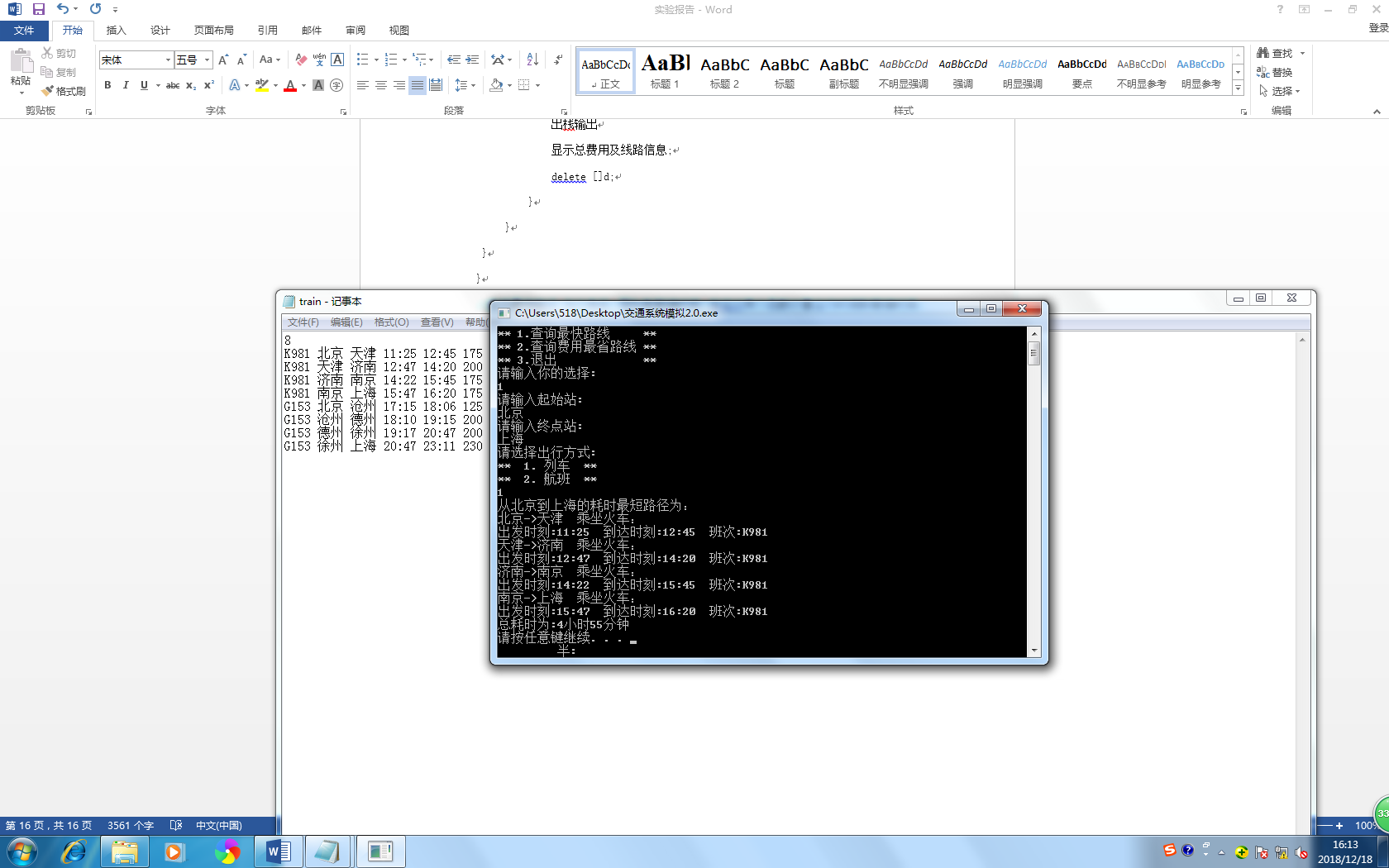
录入的时间分钟部分小于10时只显示一位，解决：输出前判断分钟部分是否小于0，若小于，则预先输出“0”。

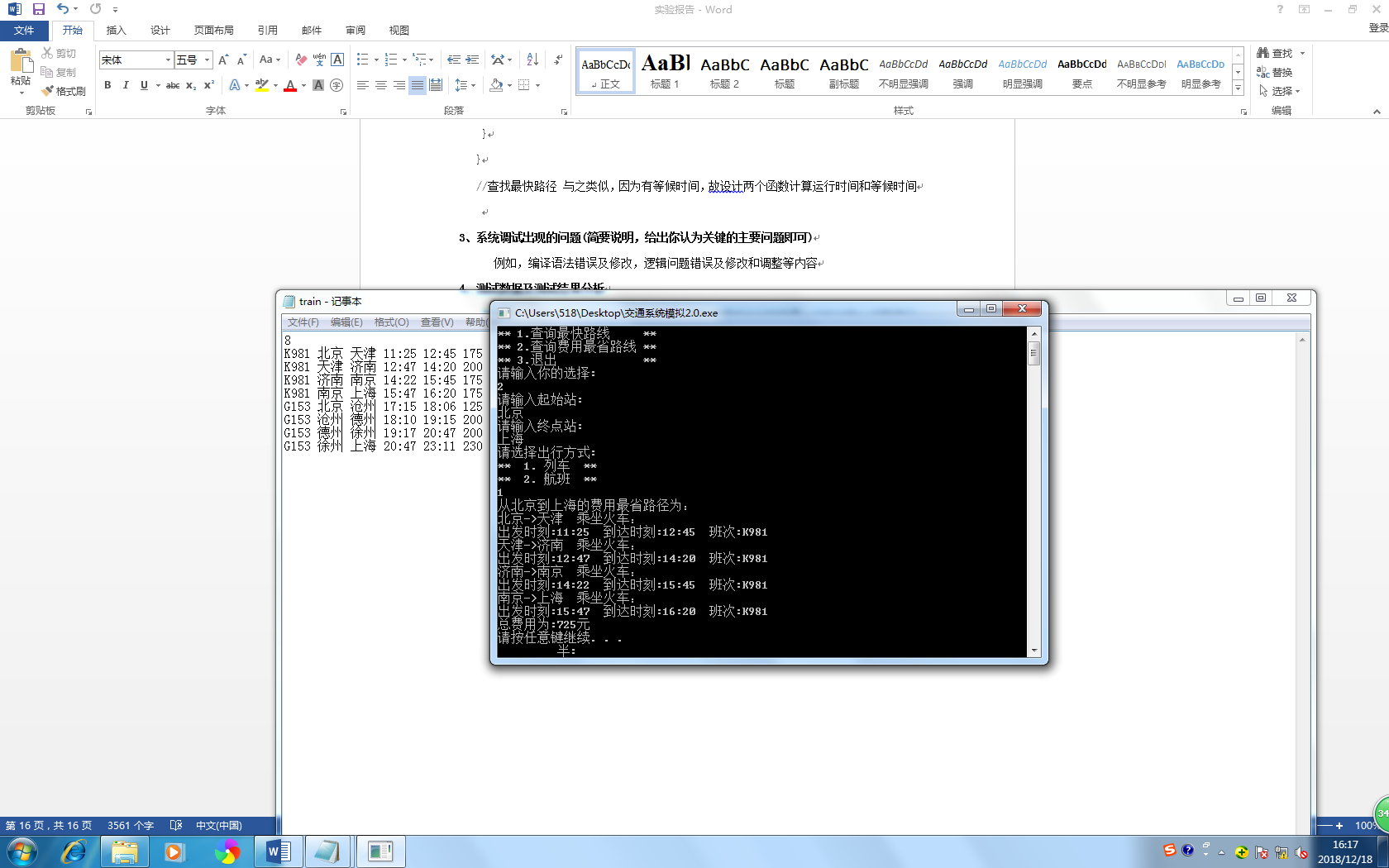
管理员操作时查看线路信息时只能查看最新录入的一条，原因：return true;置于for循环内部，导致循环提前结束，将其置于for循环外部即可。

文件读取操作时fprintf的参数只能为char类型，但结构体中定义的城市名变量为string类型，需要进行如下造作进行转换：const char \*str=AddList[i].cityname.data();

## 4、测试数据及测试结果分析

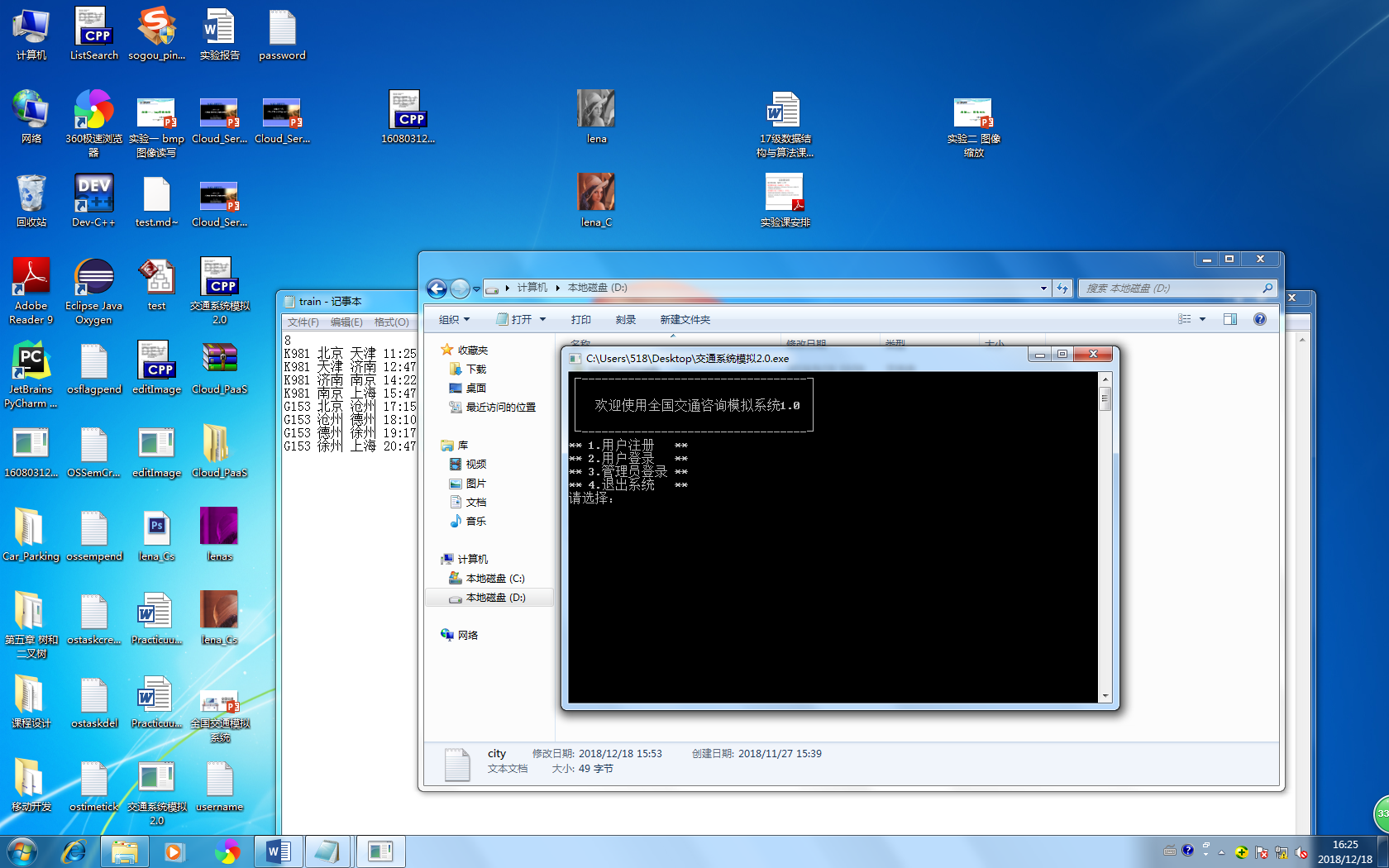
列出对于给定的输入所产生的输出结果,即程序测试运行的结果，应给出输入的数据及程序运行的结果

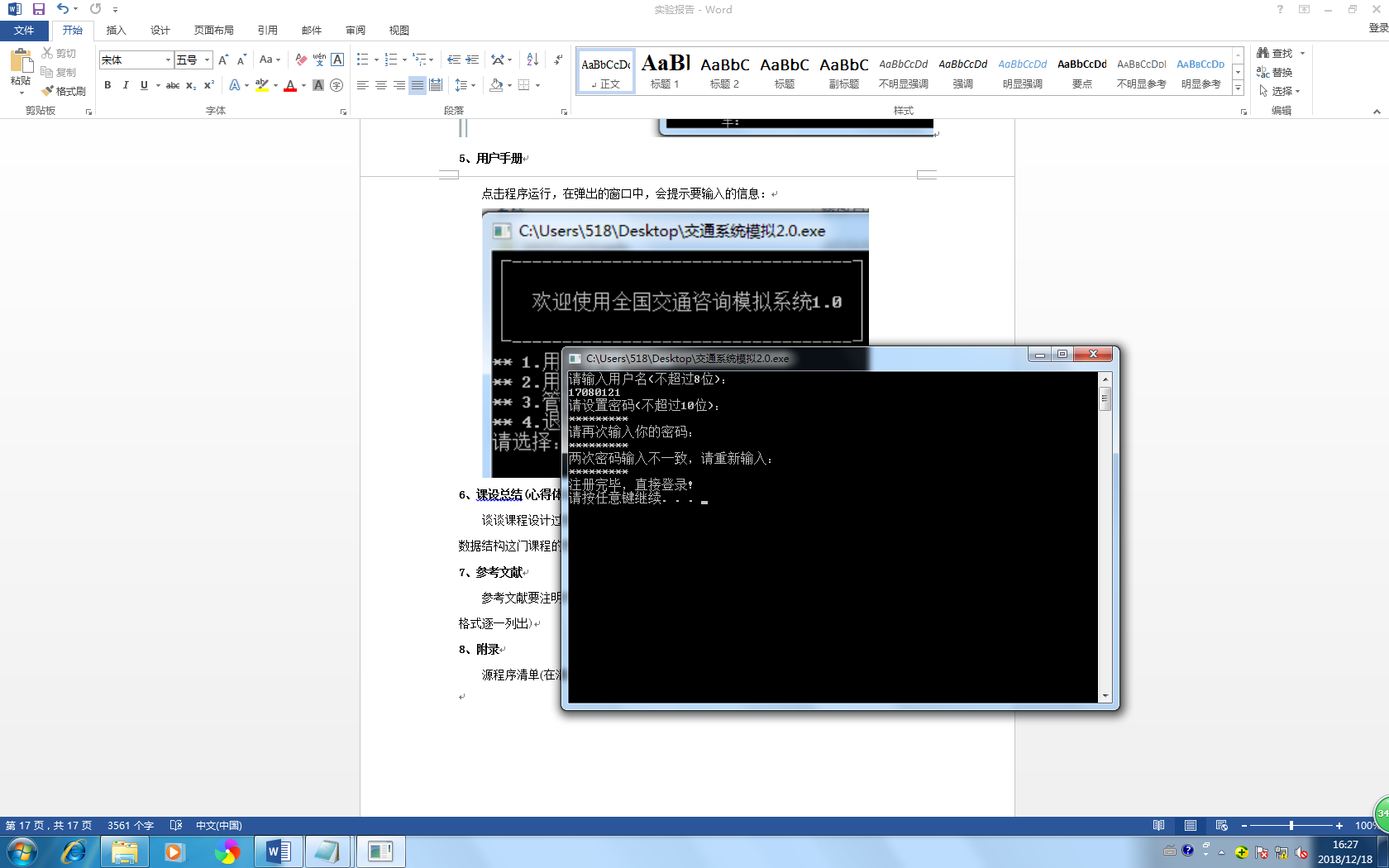




## 5、用户手册

点击程序运行，在弹出的窗口中，会提示要输入的信息：





## 6、课设总结(心得体会)

**通过这次课程设计，让我巩固了图算法尤其是迪杰斯特拉算法的设计，并在线路的输出中灵活的使用栈的数据结构进行操作，再删除表结点的时候将表尾系结点替换删除结点的方法降低了算法的时间复杂度，在文件操作中以const char file[]在D盘动态创建了三个存储文件，并能实现先在文件输入数据，程序可正常读取运行。总体上让我的程序设计和调试能力有所提升，也使我认识到了我在程序设计方面的不足，以及专业知识的匮乏，也使得我在课后广泛的查阅资料进行学习。我的程序设计思想和代码的编写和调试能力在此次课程设计期间有所提高。这样一种更贴近实际的课程设计，大大地提高了我的学习效率，希望能够在平时教学中也能加入一些类似的课程设计的程序设计题。**

## 7、参考文献

参考文献要注明作者、出版社、出版日期。(按“序号 作者，书名，出版社，出版时间”格式逐一列出)

**1.殷人昆 数据结构（用面向对象方法与C++语言描述） 清华大学出版社 2007.6**

## 8、附录

源程序清单(在清单中按系统功能模块或程序功能模块给出注释)

//交通系统模拟

#include<iostream>

#include<string.h>

#include<conio.h>

#include<fstream>

using namespace std;

//文件操作,在D盘建立三个文本文件

const char CityFile[] ="D:\\city.txt";

const char TrainFile[] ="D:\\train.txt";

const char FlightFile[] ="D:\\flight.txt";

const int MAX\_VERTEX\_NUM=34; //图中顶点数目最大值

const int MaxSize=20;

const int maxValue=9999;

typedef struct TrafficNodeData //交通工具信息

{

char name[MaxSize]; //班次

int StartTime; //出发时间

int StopTime; //到达时间

int Duration; //时间段

int EndCity; //边的另一顶点，即下一个城市编号

int Cost; //票价

struct TrafficNodeData \*link; //下一条边链指针

}TrafficNode;

typedef struct VNodeData //路线信息

{

string cityname ; //城市名称

int TrainNum,FlightNum; //列车、航班数量

TrafficNode \*Tadj; //列车线

TrafficNode \*Fadj; //航班线

}VNode;

//交通系统类的定义

class TrafficGraph

{

public:

TrafficGraph(); //构造函数

~TrafficGraph(); //析构函数

int InitialInterface(); //开始及登录界面

void ShowMenu(int n); //功能选择界面

bool ShowFpath(string name);//显示航班线

bool ShowTpath(string name);//显示列车线

bool DelFlight();//删除航班线

bool InsertFlight(int i,int j,int TFcost,int time1,int time2,char \*TFname);//添加航班线

bool DelTrain();//删除火车线

bool InsertTrain(int i,int j,int TFcost,int time1,int time2,char \*TFname);//添加火车线

bool DelCity(string Name);//删除城市

void InsertCity(string Name);//添加城市

int FindCity (string name);//查找城市并返回城市序号

//void InitData();

void FastestRoute(); //耗时最短路径

void CheapestRoute(); //花钱最少路径

int getCost(int v,int u,int choose); //返回两个城市之间的票价

int getDuration(int v1,int v2,int choose); //返回行驶的时间间隔

int getTimes(int v1,int u,int k,int choose); //返回等候的时间

void getInfor(int v1,int v2,int choose); //获取列车或航班信息

int SaveSysInfo(); //将信息存入文件

int InitSysData () ; //用文件信息初始化

private:

VNode \*AddList; //系统信息

int Cnum; // 城市个数

};

TrafficGraph::TrafficGraph() //构造函数

{

AddList=new VNode[MAX\_VERTEX\_NUM];

Cnum=0;

for(int i=0;i<MAX\_VERTEX\_NUM;i++)

{

AddList[i].Tadj=NULL; //列车线指针置空

AddList[i].Fadj=NULL; //航班线指针置空

}

}

TrafficGraph::~TrafficGraph() //析构函数

{

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

{

TrafficNode \*p=AddList[i].Tadj;

while(p!=NULL)

{

AddList[i].Tadj=p->link;

delete p;p=AddList[i].Tadj;

}

TrafficNode \*q=AddList[i].Fadj;

while(q!=NULL)

{

AddList[i].Fadj=q->link;

delete q;q=AddList[i].Fadj;

}

}

delete []AddList;

}

int TrafficGraph::InitialInterface() //初始化及登录界面

{

string num1,num2;

char a[10],c[10];

char b[10];

int flag; //循环终止标记

char ab='n'; //退出、确认字符

ofstream out; //文件写入

ifstream in; //文件读出

cout<<"┌----------------------------------┐"<<endl;

cout<<"│ │"<<endl;

cout<<"│ 欢迎使用全国交通咨询模拟系统1.0 │"<<endl;

cout<<"│ │"<<endl;

cout<<"└----------------------------------┘"<<endl;

string Magname="17080207"; //管理员账号，唯一

char password[]="123456"; //管理员密码

cout<<"\*\* 1.用户注册 \*\*"<<endl;

cout<<"\*\* 2.用户登录 \*\*"<<endl;

cout<<"\*\* 3.管理员登录 \*\*"<<endl;

cout<<"\*\* 4.退出系统 \*\*"<<endl;

cout<<"请选择：";

int x;

cin>>x;

system("cls");

switch(x)

{

case 1:

cout<<"请输入用户名(不超过8位)："<<endl;

cin>>num1;

cout<<"请设置密码(不超过10位)："<<endl;

for(int j=0;j<10;j++) //密码暗码

{

a[j]=getch();

cout<<"\*"; //设置暗码

if(a[j]=='\r')//按回车

{

a[j]='\0';//密码输入结束

break;

}

}

cout<<endl<<"请再次输入你的密码："<<endl;

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

{

c[j]=getch();

cout<<"\*";

if(c[j]=='\r')

{

c[j]='\0';

break;

}

}

while(strcmp(a,c)!=0)

{

cout<<endl<<"两次密码输入不一致，请重新输入："<<endl;

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

{

c[j]=getch();

cout<<"\*";

if(c[j]=='\r')

{

c[j]='\0';

break;

}

}

}

out.open("username.txt",ios::app); //用户名存入文件

out<<num1<<endl;

out.close();

out.open("password.txt",ios::app); //对应的密码存入文件

out<<c<<endl;

out.close();

cout<<endl<<"注册完毕，直接登录!"<<endl;

x=2; //返回2，输出用户功能表

system("pause");

system("cls");

break;

case 2:

char ch[100];

int i,f;

do

{

i=1;f=0; //i标记是第几个用户，f为查找标记，找到记为0

cout<<"请输入用户名："<<endl;

cin>>num2;

in.open("username.txt");

if(!in)

{

cout<<"没有用户注册!"<<endl;

exit(1);

}

do

{

in.getline(ch,100,'\n'); //在文件中查找该用户是否注册了

if(ch==num2)

{

f=1; //是注册用户，f=1

break;

}

i++; //记录是第几个用户

}while(ch!=num2&&!in.eof());

if(f==0) //未注册

{

cout<<"该用户未注册!"<<endl;

flag=1;

in.close();

}

else

{

cout<<"请输入密码:"<<endl;

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

{

b[j]=getch();

cout<<"\*";

if(b[j]=='\r')

{

b[j]='\0';

break;

}

}

in.close();

in.open("password.txt");

for(int k=0;k<i;k++) //看密码是否正确

{

in.getline(ch,100,'\n');

}

in.close();

if(strcmp(ch,b)==0)

flag=0; //查找成功，循环标志置0

else

{

cout<<endl<<"用户名或密码输入错误！"<<endl;

flag=1; //失败，循环标志置1

system("pause");//暂停界面

system("cls"); //清屏

}

}

}while(flag==1);

cout<<endl<<"登录成功!"<<endl;

system("pause");

system("cls");

break;

case 3:

do

{

cout<<"请输入用户名："<<endl;

cin>>num2;

cout<<"请输入密码:"<<endl;

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

{

b[j]=getch();

cout<<"\*";

if(b[j]=='\r')

{

b[j]='\0';

break;

}

}

cout<<endl;

if(num2==Magname&&strcmp(password,b)==0)

flag=0; //查找成功，循环标志置0

else

{

cout<<"用户名或密码输入错误！"<<endl;

flag=1;

system("pause");

system("cls");

}

}while(flag==1); //失败，循环标志置1

cout<<endl<<"登录成功!"<<endl;

system("pause");

system("cls");

break;

case 4:

cout<<"确认退出？(y/n)"<<endl;

cin>>ab;

if(ab=='y'||ab=='Y')

exit(1); //退出程序

else

{

system("cls");

x=InitialInterface();//返回主界面

}

break;

default:

cout<<"输入非法！请重新选择："<<endl;

system("pause") ;

system("cls");

x=InitialInterface();

break;

}

return x;

}

//输出功能表

void TrafficGraph::ShowMenu(int n)

{

switch(n)

{

case 2: //用户登录

cout<<"\*\* 1.查询最快路线 \*\*"<<endl;

cout<<"\*\* 2.查询费用最省路线 \*\*"<<endl;

cout<<"\*\* 3.退出 \*\*"<<endl;

cout<<"请输入你的选择:"<<endl;

break;

case 3: //管理员登录

cout<<"\*\* 1.添加城市 \*\*"<<endl;

cout<<"\*\* 2.删除城市 \*\*"<<endl;

cout<<"\*\* 3.添加线路 \*\*"<<endl;

cout<<"\*\* 4.删除线路 \*\*"<<endl;

cout<<"\*\* 5.显示线路 \*\*"<<endl;

cout<<"\*\* 6.退出 \*\*"<<endl;

cout<<"请输入你的选择:"<<endl;

break;

default:

cout<<"没有!!"<<endl;

break;

}

}

//添加城市

void TrafficGraph::InsertCity(string Name)

{

if(Cnum<MAX\_VERTEX\_NUM)

{

AddList[Cnum].cityname=Name;

AddList[Cnum].FlightNum=0;

AddList[Cnum].TrainNum=0;

Cnum++;

cout<<"添加成功!"<<endl;

}

else

cout<<"城市已满!"<<endl;

}

//查找城市并返回城市序号：

int TrafficGraph::FindCity (string name)

{

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

{

if (name==AddList[i].cityname)

{

return i;

}

}

return -1;

}

//删除城市

bool TrafficGraph::DelCity(string Name)

{

int CityNo; //城市序号

CityNo=FindCity(Name);

if(Cnum==1||CityNo<0||CityNo>=MAX\_VERTEX\_NUM) return false; //表空或顶点号超出范围

TrafficNode \*p,\*s,\*t;int i,k;

while(AddList[CityNo].Tadj!=NULL) //删除第CityNo个边链表中所有结点

{

p=AddList[CityNo].Tadj;k=p->EndCity; //取邻接顶点k

s=AddList[k].Tadj;t=NULL; //找对称存在的边结点

while(s!=NULL&&s->EndCity!=CityNo)

{

t=s;s=s->link;

}

if(s!=NULL) //删除对称存在的边结点

{

if(t==NULL)AddList[k].Tadj=s->link;

else t->link=s->link;

delete s;

}

AddList[CityNo].Tadj=p->link; //清除顶点v的边链表结点

delete p;AddList[CityNo].TrainNum--; //边数减一

}

while(AddList[CityNo].Fadj!=NULL) //航班线同上

{

p=AddList[CityNo].Fadj;k=p->EndCity;

s=AddList[k].Fadj;t=NULL;

while(s!=NULL&&s->EndCity!=CityNo)

{

t=s;s=s->link;

}

if(s!=NULL)

{

if(t==NULL)AddList[k].Fadj=s->link;

else t->link=s->link;

delete s;

}

AddList[CityNo].Fadj=p->link;

delete p;AddList[CityNo].FlightNum--;

}

Cnum--; //城市数减一

AddList[CityNo].cityname=AddList[Cnum].cityname;//填补，这里直接采用表中最后一个城市的信息进行填补

p=AddList[CityNo].Tadj=AddList[Cnum].Tadj;

t=AddList[CityNo].Fadj=AddList[Cnum].Fadj;

while(p!=NULL)//填补城市的列车线边链表非空

{

s=AddList[p->EndCity].Tadj;

while(s!=NULL)//未遍历到该班次的终点站

{

if(s->EndCity==Cnum)

{//遍历到该班次的终点站

s->EndCity=CityNo;//记下该站下标

break;

}

else s=s->link;

}

p=p->link; //修改

}

while(t!=NULL)//填补城市的航班线边链表非空

{

s=AddList[t->EndCity].Fadj;

while(s!=NULL)//未遍历到该班次的终点站

{

if(s->EndCity==Cnum)

{//遍历到该班次的终点站

s->EndCity=CityNo;//记下该站下标

break;

}

else s=s->link;

}

t=t->link;

}

return true;

}

//添加火车线

bool TrafficGraph::InsertTrain(int i,int j,int TFcost,int time1,int time2,char \*TFname)

{

if(i>=0&&i<Cnum&&j>=0&&j<Cnum)

{

TrafficNode \*p=AddList[i].Tadj; //i对应的列车线

while(p!=NULL&&p->EndCity!=j) //寻找邻接顶点j

p=p->link;

if(p!=NULL)return false; //j已存在，不插入

p=new TrafficNode; //否则，创建新结点

p->EndCity=j;p->Cost=TFcost; //链入i的列车线

strcpy(p->name,TFname);

p->StartTime=time1;//stHour\*60+stMinute;

p->StopTime=time2; //enHour\*60+enMinute;

p->Duration=time2-time1;

p->link=AddList[i].Tadj;//将该线路链入总表

AddList[i].Tadj=p;

AddList[i].TrainNum++;

return true;

}

return false;

}

//删除火车线

bool TrafficGraph::DelTrain()

{

string TSCname;

char Tno[MaxSize];

cout<<"请输入要删除的线路的始发站:"<<endl;

cin>>TSCname;

int v1,v2=-1;

v1=FindCity(TSCname);

if(ShowTpath(TSCname)) //输出以TSCname为始发站的线路

{

cout<<"请输入你想删除的班次:"<<endl;

cin>>Tno;

TrafficNode \*p=AddList[v1].Tadj;

while(p!=NULL)

{

if(strcmp(p->name,Tno)==0) //按照班次名找到对应的线路

{//找到对应的线路

v2=p->EndCity;//记录终点站线路下标

break;

}

p=p->link;

}

if(v1!=-1&&v2!=-1)

{//起始站和终点站存在

TrafficNode \*p=AddList[v1].Tadj,\*q=NULL,\*s=p;

while(p!=NULL&&p->EndCity!=v2)

{

q=p; p=p->link;

}

if(p!=NULL)

{

if(p==s) AddList[v1].Tadj=p->link;

else q->link=p->link;

delete p;

}

AddList[v1].TrainNum--;

return true;

}

return false;

}

return false;

}

//添加航班线

bool TrafficGraph::InsertFlight(int i,int j,int TFcost,int time1,int time2,char \*TFname)

{

if(i>=0&&i<Cnum&&j>=0&&j<Cnum)

{

TrafficNode \*p=AddList[i].Fadj;

while(p!=NULL&&p->EndCity!=j)

p=p->link;

if(p!=NULL)return false;

p=new TrafficNode;

p->EndCity=j;p->Cost=TFcost;

strcpy(p->name,TFname);

p->StartTime=time1;//stHour\*60+stMinute;

p->StopTime=time2;//enHour\*60+enMinute;

p->Duration=time2-time1;

p->link=AddList[i].Fadj;

AddList[i].Fadj=p;

AddList[i].FlightNum++;

return true;

}

return false;

}

//删除航班线

bool TrafficGraph::DelFlight()

{

string FSCname;

char Fno[MaxSize];

cout<<"请输入要删除的线路的始发站:"<<endl;

cin>>FSCname;

int v1,v2=-1;

v1=FindCity(FSCname);

if(ShowFpath(FSCname))

{

cout<<"请输入你想删除的班次:"<<endl;

cin>>Fno;

TrafficNode \*p=AddList[v1].Fadj;

while(p!=NULL)

{

if(strcmp(p->name,Fno)==0)

{

v2=p->EndCity;

break;

}

p=p->link;

}

if(v1!=-1&&v2!=-1)

{

TrafficNode \*p=AddList[v1].Fadj,\*q=NULL,\*s=p;

while(p!=NULL&&p->EndCity!=v2)

{

q=p; p=p->link;

}

if(p!=NULL)

{

if(p==s) AddList[v1].Fadj=p->link;

else q->link=p->link;

delete p;

}

AddList[v2].FlightNum--;

return true;

}

return false;

}

return false;

}

//显示列车线

bool TrafficGraph::ShowTpath(string name)

{

int i;

i=FindCity(name);

if(i!=-1)

{

if(AddList[i].Tadj!=NULL)

{

cout<<"班次编号\t"<<"终点站\t\t"<<"出发时间\t"<<"到达时间\t"<<"票价\t"<<endl;

for(TrafficNode \*p=AddList[i].Tadj;p!=NULL;p=p->link)//循环搜索班次编号

{

cout<<p->name<<"\t\t";

cout<<AddList[p->EndCity].cityname<<"\t\t";

cout<<p->StartTime/60<<":";

if(p->StartTime%60<10)

{

cout<<"0";

}

cout<<p->StartTime%60<<"\t\t";

cout<<p->StopTime/60<<":";

if(p->StopTime%60<10)

{

cout<<"0";

}

cout<<p->StopTime%60<<"\t\t";

cout<<p->Cost<<"\t\t";

cout<<endl;

}

return true;

}

else

{

cout<<"未添加线路!"<<endl;

return false;

}

}

else

{

cout<<"目标城市未录入!"<<endl;

return false;

}

}

//显示航班线

bool TrafficGraph::ShowFpath(string name)

{

int i;

i=FindCity(name);

if(i!=-1)

{

if(AddList[i].Fadj!=NULL)

{

cout<<"班次编号\t"<<"终点站\t\t"<<"出发时间\t"<<"到达时间\t"<<"票价\t"<<endl;

for(TrafficNode \*p=AddList[i].Fadj;p!=NULL;p=p->link)//循环搜索班次编号

{

cout<<p->name<<"\t\t";

cout<<AddList[p->EndCity].cityname<<"\t\t";

cout<<p->StartTime/60<<":";

if(p->StartTime%60<10)

{

cout<<"0";

}

cout<<p->StartTime%60<<"\t\t";

cout<<p->StopTime/60<<":";

if(p->StopTime%60<10)

{

cout<<"0";

}

cout<<p->StopTime%60<<"\t\t";

cout<<p->Cost<<"\t\t";

cout<<endl;

}

return true;

}

else

{

cout<<"未添加线路!"<<endl;

return false;

}

}

else

{

cout<<"目标城市未录入!"<<endl;

return false;

}

}

//获取两个城市间的费用

int TrafficGraph::getCost(int v,int u,int choose)

{

if(v!=-1&&u!=-1)

{

if(v==u) return 0;

TrafficNode \*p=new TrafficNode;

switch(choose)

{

case 1:

p=AddList[v].Tadj;

break;

case 2:

p=AddList[v].Fadj;

break;

}

while(p!=NULL&&p->EndCity!=u)

p=p->link;

if(p!=NULL)return p->Cost;

else return maxValue;

}

return 0;

}

// 获取线路信息

void TrafficGraph::getInfor (int v1,int v2,int choose)

{

if(v1!=-1&&v2!=-1)

{

TrafficNode \*p=new TrafficNode;

switch(choose)

{

case 1:

cout<<"乘坐火车："<<endl;

p=AddList[v1].Tadj;

break;

case 2:

cout<<"乘坐飞机："<<endl;

p=AddList[v1].Fadj;

break;

}

while(p!=NULL&&p->EndCity!=v2)

p=p->link;

if(p!=NULL)

{

cout<<"出发时刻:"<<p->StartTime/60<<":";

if(p->StartTime%60<10)

cout<<"0";

cout<<p->StartTime%60<<"\t";

cout<<"到达时刻:"<<p->StopTime/60<<":";

if(p->StopTime%60<10)

cout<<"0";

cout<<p->StopTime%60<<"\t";

cout<<"班次:"<<p->name<<endl;

}

}

}

//查找最便宜路线

void TrafficGraph::CheapestRoute()

{

int n=Cnum;

int choose;

bool \*S=new bool[n]; //最短路径顶点集

int i,j,k,w,min;

int dist[n];

//dist[i]是当前求到的从顶点v到顶点i的最短路径长度

int path[n];

//path[i]存放求到的最短路径

string StartCity,EndCity;

cout<<"请输入起始站:"<<endl;

cin>>StartCity;

cout<<"请输入终点站:"<<endl;

cin>>EndCity;

int v1=FindCity(StartCity) ;

int v2=FindCity(EndCity);

if(v1!=-1&&v2!=-1)

{

cout<<"请选择出行方式:"<<endl;

cout<<"\*\* 1. 列车 \*\*"<<endl;

cout<<"\*\* 2. 航班 \*\*"<<endl;

cin>>choose;

if(choose==1||choose==2)

{

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

{

dist[i]=getCost(v1,i,choose); //权重数组初始化

S[i]=false; //顶点集数组初始化

if(i!=v1&&dist[i]<maxValue) path[i]=v1;//记下该顶点下标

else path[i]=-1;

}

S[v1]=true; dist[v1]=0; //顶点v加入顶点集合

for(i=0;i<n-1;i++) //小于除始发站的城市个数

{

min=maxValue; int u=v1; //选不在S中具有最短路径的顶点u

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

if(S[j]==false&&dist[j]<min)

{//j不在s中且路径长度小于最小值

u=j;min=dist[j];

}

S[u]=true; //将顶点u加入集合S

for(k=0;k<n;k++){ //修改

w=getCost(u,k,choose);

if(S[k]==false&&w<maxValue&&dist[u]+w<dist[k]){

//顶点k未加入S，且绕过u可以缩短路径

dist[k]=dist[u]+w;

path[k]=u; //修改到k的最短路径

}

}

}

if(dist[v2]<maxValue)

{

cout<<"从"<<AddList[v1].cityname<<"到"<<AddList[v2].cityname<<"的费用最省路径为："<<endl;

int \*d=new int [MAX\_VERTEX\_NUM]; //栈数组

int top=-1;

j=v2;

while(j!=v1)

{

top++;

d[top]=j;

j=path[j];

}

cout<<AddList[v1].cityname;

d[top+1]=v1;

while(top!=-1){

cout<<"->"<<AddList[d[top]].cityname<<" ";

getInfor(d[top+1],d[top],choose);

top--;

if(top!=-1)

{

cout<<AddList[d[top+1]].cityname;

}

}

cout<<"总费用为:"<<dist[v2]<<"元"<<endl;

delete []d;

}

else

cout<<"未找到费用最省路径！"<<endl;

}

else

cout<<"选择方式输入有误!"<<endl;

}

else

cout<<"目的地或出发地不存在!"<<endl;

}

//获取时间段

int TrafficGraph::getDuration(int v1,int v2,int choose)

{

if(v1!=-1&&v2!=-1)

{

if(v1==v2) return 0;

TrafficNode \*p=new TrafficNode;

switch(choose)

{

case 1:

p=AddList[v1].Tadj;

break;

case 2:

p=AddList[v1].Fadj;

break;

}

while(p!=NULL&&p->EndCity!=v2)

p=p->link;

if(p!=NULL)return p->Duration;

else return maxValue;

}

return 0;

}

//获取等候时间

int TrafficGraph::getTimes(int v1,int u,int k,int choose)

{

if(v1!=-1&&u!=-1&&k!=-1)

{

TrafficNode \*p=new TrafficNode,\*q=new TrafficNode;

switch(choose)

{

case 1:

p=AddList[v1].Tadj;

q=AddList[u].Tadj;

break;

case 2:

p=AddList[v1].Fadj;

q=AddList[u].Fadj;

break;

}

while(p!=NULL&&p->EndCity!=u)

{

p=p->link;

}

while(q!=NULL&&q->EndCity!=k)

{

q=q->link;

}

if(p!=NULL&&q!=NULL)

{

return (q->StartTime-p->StopTime);

}

else

return maxValue;

}

return 0;

}

//查找最快路径

void TrafficGraph::FastestRoute()

{

int n=Cnum;

int choose;

bool \*S=new bool[n]; //最省时路径顶点集

int i,j,k,w,x,min;

int dist[n];

int path[n];

string StartCity,EndCity;

cout<<"请输入起始站:"<<endl;

cin>>StartCity;

cout<<"请输入终点站:"<<endl;

cin>>EndCity;

int v1=FindCity(StartCity);

int v2=FindCity(EndCity);

if(v1!=-1&&v2!=-1)

{

cout<<"请选择出行方式:"<<endl;

cout<<"\*\* 1. 列车 \*\*"<<endl;

cout<<"\*\* 2. 航班 \*\*"<<endl;

cin>>choose;

if(choose==1||choose==2)

{

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

{

dist[i]=getDuration(v1,i,choose); //数组初始化

S[i]=false;

if(i!=v1&&dist[i]<maxValue) path[i]=v1;

else path[i]=-1;

}

S[v1]=true; dist[v1]=0; //顶点v加入顶点集合

for(i=0;i<n-1;i++)

{

min=maxValue; int u=v1; //选不在S中具有最短路径的点顶u

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

if(S[j]==false&&dist[j]<min)

{

u=j;min=dist[j];

}

S[u]=true; //将顶点u加入集合S

for(k=0;k<n;k++){ //修改

w=getDuration(u,k,choose);

x=getTimes(path[u],u,k,choose);

//if(w<maxValue)

//cout<<"x="<<x<<endl;

if(S[k]==false&&w<maxValue&&x>0&&dist[u]+w+x<dist[k]){

//顶点k未加入S，且绕过u可以缩短路径

dist[k]=dist[u]+w+x;

path[k]=u; //修改到k的最短路径

}

}

}

if(dist[v2]<maxValue)

{

cout<<"从"<<AddList[v1].cityname<<"到"<<AddList[v2].cityname<<"的耗时最短路径为："<<endl;

int \*d=new int [MAX\_VERTEX\_NUM]; //栈数组

int top=-1;

j=v2;

while(j!=v1)

{

top++;

d[top]=j;

j=path[j];

}

d[top+1]=v1;

cout<<AddList[v1].cityname;

while(top!=-1){

cout<<"->"<<AddList[d[top]].cityname<<" ";

getInfor(d[top+1],d[top],choose);

top--;

if(top!=-1)

{

cout<<AddList[d[top+1]].cityname;

}

}

cout<<"总耗时为:"<<dist[v2]/60<<"小时"<<dist[v2]%60<<"分钟"<<endl;

delete []d;

}

else

cout<<"未找耗时最短路径！"<<endl;

}

else

cout<<"选择方式输入有误!"<<endl;

}

else

cout<<"目的地或出发地不存在！"<<endl;

}

//将交通系统信息写入文件

int TrafficGraph::SaveSysInfo()

{

FILE \*fp;

int i,j,total;

TrafficNode \*p;

fp=fopen(CityFile,"w"); //打开城市文件

fprintf(fp,"%d\n",Cnum); //读取文件中城市个数

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

{ //依次读取各城市名

const char \*str=AddList[i].cityname.data(); //string类型转char

fprintf(fp,"%s\n",str);

}

fclose(fp);total=0;

fp=fopen(TrainFile,"w"); //打开列车文件

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

{

total+=AddList[i].TrainNum;

}

fprintf(fp,"%d\n",total);

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

{

for (p=AddList[i].Tadj;p!=NULL;p=p->link)

{

const char\*a=AddList[i].cityname.data(); //string类型转char

const char\*b=AddList[p->EndCity].cityname.data(); //string类型转char

fprintf(fp,"%s %s %s ", AddList[i].Tadj->name, a, b);

fprintf(fp,"%2d:%2d %2d:%2d %d\n", p->StartTime/60,

p->StartTime%60, p->StopTime/60, p->StopTime%60, p->Cost);

}

}

fclose(fp);total=0;

fp=fopen(FlightFile,"w");

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

{

total+=AddList[i].FlightNum;

}

fprintf(fp,"%d\n",total);

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

{

for (p=AddList[i].Fadj;p!=NULL;p=p->link)

{

const char\*a=AddList[i].cityname.data();

const char\*b=AddList[p->EndCity].cityname.data();

fprintf(fp,"%s %s %s ", AddList[i].Fadj->name, a, b);

fprintf(fp,"%2d:%2d %2d:%2d %d\n", p->StartTime/60,

p->StartTime%60, p->StopTime/60, p->StopTime%60, p->Cost);

}

}

fclose(fp);

return 1;

}

//用文件信息初始化系统

int TrafficGraph::InitSysData()

{

FILE \*fp;

TrafficNode \*p;

char \*a,\*b,\*c;

a=new char;

b=new char;

c=new char;

int i,j,k,hour1,minute1,num,cost,hour2,minute2;

fp=fopen(CityFile,"r"); //打开城市文件

if (!fp)

{

printf("\nError:Cannot Open City File...\n");

return -1;

}

fscanf(fp,"%d",&Cnum);

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

{

char \*p;

p=new char;

fscanf(fp,"%s",p); //读取城市名

AddList[i].cityname=p;

}

fclose(fp);

fp=fopen(TrainFile,"r");//打开列车文件

if (!fp)

{

printf("\nError:Cannot Open Train File...\n");

return -1;

}

fscanf(fp,"%d",&num);

for (i=0;i<num;i++) //根据文件中列车班次总数进行读取

{

fscanf(fp,"%s",a);//读取列车班次

fscanf(fp,"%s",b);//读取始发站

fscanf(fp,"%s",c);//读取终点站

j=FindCity(b); //始发站下标

k=FindCity(c);

fscanf(fp,"%d:%d",&hour1,&minute1);

fscanf(fp,"%d:%d",&hour2,&minute2);

fscanf(fp,"%d",&cost);

InsertTrain(j,k,cost,hour1\*60+minute1,hour2\*60+minute2,a);

}

fclose(fp);

fp=fopen(FlightFile,"r");//打开航班文件

if (!fp)

{

printf("\nError:Cannot Open Flight File...\n");

return -1;

}

fscanf(fp,"%d",&num);

for (i=0;i<num;i++) //根据文件中航班班次总数进行读取

{

fscanf(fp,"%s",a); //读取航班班次

fscanf(fp,"%s",b); //读取始发站

fscanf(fp,"%s",c); //读取终点站

j=FindCity(b); //始发站下标

k=FindCity(c);

fscanf(fp,"%d:%d",&hour1,&minute1);

fscanf(fp,"%d:%d",&hour2,&minute2);

fscanf(fp,"%d",&cost);

InsertFlight(j,k,cost,hour1\*60+minute1,hour2\*60+minute2,a);

}

fclose(fp);

return 1;

}

int main()

{

int a,b,c;

char ch='y';

string name;

string StartCity,EndCity;

int TFcost;

int stHour,stMinute;

int enHour,enMinute;

char ah;

char TFname[MaxSize];

bool bo;

TrafficGraph tr;

//tr.InitSysData();

do

{

a=tr.InitialInterface();

if(a==2)

{

do

{

tr.ShowMenu(2);

cin>>b;

switch(b)

{

case 1:

tr.FastestRoute();

break;

case 2:

tr.CheapestRoute();

break;

case 3:

ch='n';

break;

default:

cout<<"输入非法！！！"<<endl;

}

system("pause");

system("cls");

}while(ch=='y');

}

else if(a==3)

{

do{

tr.ShowMenu(3);

cin>>b;

if(b>=0&&b<=9)

{

switch(b)

{

case 1:

cout<<"请输入城市名："<<endl;

cin>>name;

tr.InsertCity(name);

tr.SaveSysInfo();

break;

case 2:

cout<<"请输入要删除的城市名:"<<endl;

cin>>name;

bo=tr.DelCity(name);

if(bo) cout<<"删除成功!"<<endl;

else cout<<"删除失败!"<<endl;

tr.SaveSysInfo();

break;

case 3:

cout<<"1.列车"<<endl;

cout<<"2.航班"<<endl;

cin>>c;

cout<<"请输入起始站:"<<endl;

cin>>StartCity;

cout<<"请输入终点站:"<<endl;

cin>>EndCity;

int i,j;

i=tr.FindCity(StartCity);//始发站城市编号

j=tr.FindCity(EndCity); //终点站城市编号

if(i>=0&&j>=0)

{

cout<<"请输入花费:"<<endl;

cin>>TFcost;

cout<<"出发时刻(00:00,24小时制):"<<endl;

cin>>stHour>>ah>>stMinute;

cout<<"到达时刻(00:00,24小时制):"<<endl;

cin>>enHour>>ah>>enMinute;

while(stHour\*60+stMinute>=enHour\*60+enMinute)

{

cout<<"时间输入异常!请重新输入:"<<endl;

cout<<"出发时刻(00:00,24小时制):"<<endl;

cin>>stHour>>ah>>stMinute;

cout<<"到达时刻(00:00,24小时制):"<<endl;

cin>>enHour>>ah>>enMinute;

}

cout<<"请输入班次编号:"<<endl;

cin>>TFname;

switch(c)

{

case 1:

bo=tr.InsertTrain(i,j,TFcost,stHour\*60+stMinute,enHour\*60+enMinute,TFname);

break;

case 2:

bo=tr.InsertFlight(i,j,TFcost,stHour\*60+stMinute,enHour\*60+enMinute,TFname);

break;

default:

cout<<"输入非法！"<<endl;

break;

}

if(bo) cout<<"插入成功!"<<endl;

else cout<<"插入失败!"<<endl;

tr.SaveSysInfo();

}

else

{

cout<<"起始站或终点站不存在!"<<endl;

}

break;

case 4:

cout<<"1.列车"<<endl;

cout<<"2.航班"<<endl;

cin>>c;

switch(c)

{

case 1:

bo=tr.DelTrain();

break;

case 2:

bo=tr.DelFlight();

break;

default:

cout<<"输入非法！"<<endl;

break;

}

if(bo) cout<<"删除成功!"<<endl;

else cout<<"删除失败!"<<endl;

tr.SaveSysInfo();

break;

case 5:

cout<<"请输入要搜索的始发站:"<<endl;

cin>>name;

cout<<"1.列车"<<endl;

cout<<"2.航班"<<endl;

cin>>c;

switch(c)

{

case 1:

tr.ShowTpath(name);

break;

case 2:

tr.ShowFpath(name);

break;

default:

cout<<"输入非法！"<<endl;

break;

}

break;

case 6:

ch='n';

break;

default:

cout<<"输入非法!"<<endl;

break;

}

}

else

{

cout<<"输入错误！"<<endl;

ch='n';

}

system("pause");

system("cls");

}while(ch=='y'||ch=='Y');

}

}while(a!=4);

return 0;

}

# B-Trees 的实现及分析

【问题描述】：

B-Trees 是一类满足特殊条件的M 路查找树。设计并实现B-Trees的查找算法。

【基本要求】：

（1）实现在B-树上的查找，并分析其时间复杂性。

（2）实现B-树的ADT，包括其上的基本操作：结点的加入和删除。

（3）要求B-树结构中的M=3 或5，实现其中的一种即可。

（4）实现基本操作的演示。

【实现提示】：

主要考虑结点的分裂和合并。

## 1、设计目的与内容(需求分析)

即进行需求分析，确定每个模块的功能要求。即根据设计题目的要求，充分地分析和理解问题，明确问题要求做什么？（而不是怎么做？）

1、）主函数



2、）B-树上的查找：输入一个关键字的值，要求查找树中是否存在该值，输出结果。



3、）B-树结点的添加（注意结点的分裂）：将关键字插入B-树中



4、）B-树结点的删除（注意结点的合并）：从B-树中移除关键字。



5、）B-树输出：按照次序输出B-树的所有关键字。

## 2、算法的基本思想

进行概要设计和详细设计。说明用到的数据结构定义、主程序的流程及各程序模块之间的调用关系。并用自然语言描述每个模块所涉及的算法。

1. 概要设计（包括选择什么数据结构？数据结构采用哪种存储方式？选择的原因？设计哪些操作？这些操作之间的调用关系等等），即系统总体设计－确定总体框架，确定系统功能模块（或程序功能模块），并给出说明，用文字或框图均可；

根据题目要求，选择树作为数据结构，采用链式存储，因为节省空间且便于实现后续操作；设计的操作包括：插入结点，删除结点，查找关键字，输出树等；这些操作之间本身没有相互调用，但是要实现插入节点，还得调用分裂子节点函数和检查关键字是否存在函数；要实现删除操作，还得调用合并子节点函数。

1. 详细设计（包括数据结构的类型定义，每个操作的算法描述），在总体设计基础上设计系统总体框架、用到的库函数、关键算法的流程图；数据结构及全局变量说明。

基本操作的伪代码：

1.查找关键字函数：search()；

如果结点指针为空，返回错误；

否则，找到使key<=pNode->keyValue[i]成立的最小下标i

如果key=pNode->keyValue[i]，返回查找成功

否则，检查结点是否为叶子结点，是则返回失败；否则递归调用search()函数

2.插入结点函数：insert()；

先查看关键字是否已经存在，已存在则返回插入失败。否则，继续进行

检查是否为空树，若是，则为根结点分配内存空间。否则，继续进行

然后，检查根结点是否已满（树中每个结点至多有2\*M-1个子结点，M是树的最小度数），若是，则创建新的根结点，然后调用函数将旧的根结点分裂；否则，继续进行。

调用函数插入关键字。

3.结点删除函数：remove()；

首先需要明确一点：删除非叶子结点必然会导致不满足B树性质

那么可以这样处理：被删关键字为该结点中第i个关键字keyValue[i]，则可从指针pChild[i]所指的子树中找出最小关键字Y，代替keyValue[i]的位置，然后在叶结点中删去Y。 因此，把在非叶结点删除关键字k的问题就变成了删除叶子结点中的关键字的问题了。

首先，检查关键字是否存在，若不存在，返回失败信息；否则，继续进行。

a. 如果根结点只有一个关键字

若根结点为叶子结点，则直接清空树表

若不是叶子结点，则判断两个孩子结点关键字个数是否为最少，如果均为最少，则合并两个子节点，然后更新根结点。调用递归删除函数删除关键字.

b. 递归删除关键字：

查找关键字位置

（1）如果关键字正好在该节点中

判断它是否为叶子结点，若是，则直接删除

否则，

1）被删关键字key所在结点中前于key的子节点的关键字数目不小于CHILD\_MIN，则只需获取key的前驱关键字并替换key

2）被删关键字key所在结点中后于key的子节点的关键字数目不小于CHILD\_MIN，则只需获取key的后继关键字并替换key

3）被删关键字key所在结点中前于key的子节点和后于key的子节点的的关键字数目均等于KEY\_MIN，则合并两个结点，并递归调用删除函数。

（2）关键字key不在该节点中

如果包含key的子树根节点，关键字数目等于KEY\_MIN

1）若这时与该节点相邻的右兄弟（右兄弟没有则左兄弟）节点的关键码个数不小于CHILD\_MIN，则做调整以达到新的平衡。

2）若这时与该节点相邻的右兄弟（或左兄弟）结点关键码个数等于KEY\_MIN，则合并这两个节点。

否则，递归调用删除函数；

4.结点的分裂：splitChild（）

为分裂后的右结点分配内存空间。

将pChild(关键字已满的结点)中的后KEY\_MIN位关键字加入到新申请的pRightNode中，如果pChild为非叶子节点，则将对应的孩子节点指针也拷贝到pRightNode中。

将父亲节点中的nChildIndex后的所有关键字的值和子树指针向后移一位，为pRightNode的插入腾位置。

最后，把pChild的中间值提到父节点的nChildIndex位置。

5.结点的合并：mergeChild（）

pChild1是节点pNode中前于key的子节点

pChild2是节点pNode中后于key的子节点

将pChild2数据合并到pChild1

将父节点index的值下移

将 pChild2添加到pChild1中

父节点删除index的key，index后的往前移一位

删除pChild2

**复杂度分析**

B-树查找包含两种基本动作：

●在B-树上查找结点

●在结点中找关键字

查找的次数取决于关键字结点在B-树上的层次数。

若n≥1，m≥3，则对任意一棵具有n个关键字的m阶B-树，其树高度h至多为logt((n+1)/2)+1，t= ceil(m/2)。也就是说根结点到关键字所在结点的路径上涉及的结点数不超过logt((n+1)/2)+1。即时间复杂度为O(log(n+1))

## 3、系统调试出现的问题

B-树对结点包含的关键字和孩子结点的个数有要求，所以插入和删除时都要时刻注意这个问题；后来在插入函数和删除函数中都加了对这种特殊情况的处理：

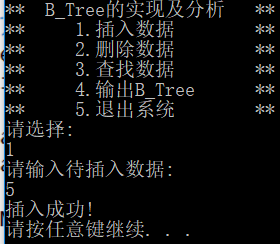
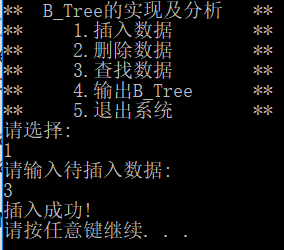
插入函数中，对根结点已满的情况，先将其分裂，再进行插入；注意分裂是将一半数量的关键字元素分裂到新的其相邻右结点中，中间关键字元素上移到父结点中（当然，如果父结点空间满了，也同样需要“分裂”操作），而且当结点中关键元素向右移动了，相关的指针也需要向右移。如果在根结点插入新元素，空间满了，则进行分裂操作，这样原来的根结点中的中间关键字元素向上移动到新的根结点中，因此导致树的高度增加一层。

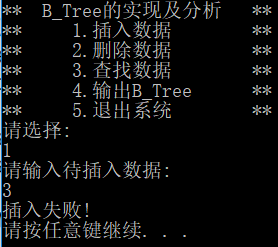
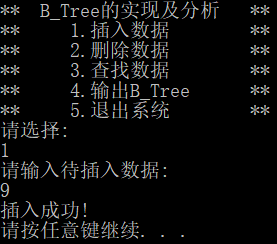
删除函数中，对根结点关键字个数为一的情况进行特殊处理。注意合并操作是首先移动父结点中的元素（该元素在两个需要合并的两个结点元素之间）下移到其子结点中，然后将这两个结点进行合并成一个结点。

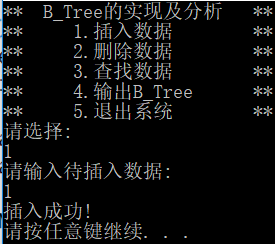
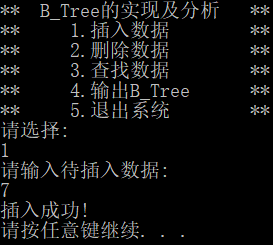
## 4、测试数据及测试结果分析

列出对于给定的输入所产生的输出结果,即程序测试运行的结果，应给出输入的数据及程序运行的结果

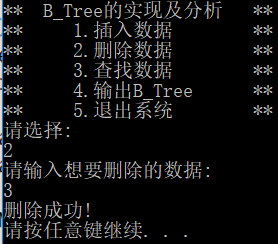
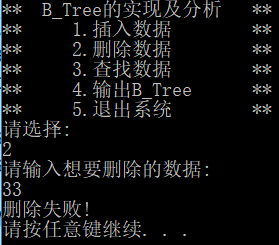
插入：

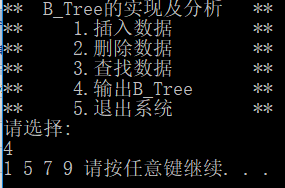


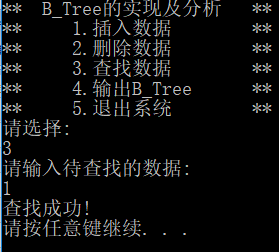
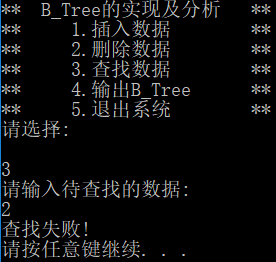




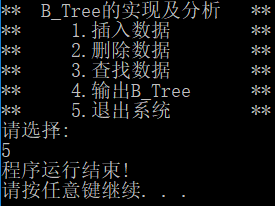
删除：



查看：  


查找某个关键字：  


退出：



## 5、用户手册

\*\* B\_Tree的实现及分析 \*\*

\*\* 1.插入数据 \*\*

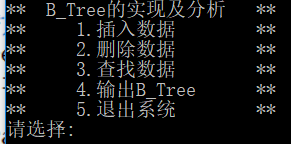
\*\* 2.删除数据 \*\*

\*\* 3.查找数据 \*\*

\*\* 4.输出B\_Tree \*\*

\*\* 5.退出系统 \*\*

请选择：



## 6、课设总结(心得体会)

谈谈课程设计过程的收获、遇到问题及解决问题过程的思考、程序调试能力的思考、对数据结构这门课程的思考、在课程设计过程中对《数据结构》课程的认识等内容。

做这个课设题目的过程中确实收获了许多，我们熟悉的树的结构是二叉树，例如完全二叉树、哈夫曼树、平衡二叉树、二叉查找树等，这些树都是针对于内存中的数据操作，它们每个结点最多只有两个孩子，当数据量大时(结点数目很多)，就会导致树很高。而B-树的一个很重要的特征就是高度小，实现方法就是让每个结点可以拥有多个(远远大于2)孩子，但是，为了在插入、删除中仍然保持B树的性质(比如高度要低)，需要规定每个结点最多包含多少个关键字(项)，最少需要包含多少个关键字，这样在插入和删除时就需要特别注意。

顶着巨大的压力，在六级考试前两天正式开始编写我们组的第三个课设题目B-Trees 的实现及分析，结合数据结构课本以及网上找到的资料，勉强完成了这一功能简单、实现却超级复杂的B-Tree; B树中结点的插入会涉及到结点的分裂问题，结点的删除会涉及到结点的合并，这是本程序的两大难点，在结合课本上的描述以及代码进行了数次比对阅读后，终于大致了解了运作流程，照着伪码一步步完成了代码的编写。

## 7、参考文献

参考文献要注明作者、出版社、出版日期。(按“序号 作者，书名，出版社，出版时间”格式逐一列出)

作者：殷人昆； 书名 ：数据结构（用面向对象方法与C++语言描述）第二版 ；出版社：清华大学出版社；出版时间：2007年6月第二版

## 8、附录

源程序清单(在清单中按系统功能模块或程序功能模块给出注释)

**//B-Tree的实现**

**#include<iostream>**

**using namespace std;**

**static const int M=3; //B树的最小度数（阶数为2M）**

**static const int KEY\_MAX=2\*M-1; //节点包含关键字的最大个数**

**static const int KEY\_MIN=M-1; //非根节点包含关键字的最小个数**

**static const int CHILD\_MAX=KEY\_MAX+1; //孩子节点的最大个数**

**static const int CHILD\_MIN=KEY\_MIN+1; //孩子节点的最小个数**

**typedef struct Node**

**{**

**bool isLeaf; //是否是叶子节点**

**int keyNum; //节点包含的关键字数量**

**int keyValue[KEY\_MAX]; //关键字的值数组**

**Node \*pChild[CHILD\_MAX]; //子树指针数组**

**Node(bool b=true, int n=0):isLeaf(b), keyNum(n){}**

**}Node;**

**class BTree**

**{**

**public:**

**BTree(); //构造函数**

**~BTree(); //析构函数**

**bool insert(const int &key); //向B数中插入新结点key**

**bool deletekey(const int &key); //从B中删除结点key**

**void display();//打印树的关键字**

**bool contain(const int &key); //检查该key是否存在于B树中**

**void clear(); //清空B树**

**void recursive\_delete(Node \*pNode); //删除树**

**void deleteNode(Node \*&pNode);//删除节点**

**bool search(Node \*pNode, const int &key);//查找关键字**

**void splitChild(Node \*pParent, int nChildIndex, Node \*pChild);//分裂子节点**

**void insertNonFull(Node \*pNode, const int &key);//在非满节点中插入关键字**

**void displayBtree(Node \*pNode); //打印树**

**void mergeChild(Node \*pParent, int index);//合并两个子节点**

**void recursive\_delete(Node \*pNode, const int &key);//递归的删除关键字**

**int getbefore(Node \*pNode);//找到前驱关键字**

**int getafter(Node \*pNode);//找到后继关键字**

**Node \*GetRoot(){ return Root;**

**}**

**private:**

**Node \* Root; //B树的根节点**

**};**

**BTree::BTree() //构造函数**

**{**

**Root=new Node;**

**Root = NULL; //创建一棵空的B树**

**}**

**BTree::~BTree() //析构函数**

**{**

**clear();**

**}**

**bool BTree::insert(const int &key) //向B数中插入新结点key**

**{**

**//整型参数key:关键字的值**

**if (contain(key)) //检查该关键字是否已经存在**

**{**

**return false;**

**}**

**else**

**{**

**if (Root==NULL)//检查是否为空树**

**{**

**Root=new Node();**

**}**

**if (Root->keyNum==KEY\_MAX) //检查根节点是否已满**

**{**

**Node \*pNode = new Node(); //创建新的根节点**

**pNode->isLeaf = false;**

**pNode->pChild[0] = Root;**

**splitChild(pNode, 0, Root); //分裂子节点**

**Root = pNode; //更新根节点指针**

**}**

**insertNonFull(Root, key); //插入关键字**

**return true;**

**}**

**}**

**bool BTree::deletekey(const int &key) //从B中删除结点key**

**{**

**//整型参数key:关键字的值**

**if (!search(Root, key)) //不存在**

**{**

**return false;**

**}**

**if (Root->keyNum==1)//特殊情况处理**

**{**

**if (Root->isLeaf)**

**{**

**clear(); //清空树表**

**return true;**

**}**

**else**

**{**

**Node \*pChild1 = Root->pChild[0];**

**Node \*pChild2 = Root->pChild[1];**

**if (pChild1->keyNum==KEY\_MIN&&pChild2->keyNum==KEY\_MIN)**

**{**

**mergeChild(Root, 0); //合并两个子节点**

**deleteNode(Root); //删除结点**

**Root = pChild1; //更新根结点指针**

**}**

**}**

**}**

**recursive\_delete(Root, key); //递归的删除关键字**

**return true;**

**}**

**void BTree::display() //打印树的关键字**

**{**

**if(Root!=NULL)**

**displayBtree(Root);**

**else**

**cout<<"空树，没有数据!"<<endl;**

**}**

**bool BTree::contain(const int &key) //检查该key是否存在于B树中**

**{**

**//整型参数key:关键字的值**

**return search(Root, key);**

**}**

**void BTree::clear() //清空B树**

**{**

**recursive\_delete(Root); //删除树**

**Root = NULL;**

**}**

**void BTree::recursive\_delete(Node \*pNode) //删除树**

**{**

**//Node类型的参数pNode：树结点指针**

**if (pNode!=NULL)**

**{**

**if (!pNode->isLeaf)**

**{**

**for(int i=0; i<=pNode->keyNum; ++i)**

**recursive\_delete(pNode->pChild[i]);**

**}**

**deleteNode(pNode);**

**}**

**}**

**void BTree::deleteNode(Node \*&pNode) //删除节点**

**{**

**//Node类型的参数pNode：树结点指针**

**if (pNode!=NULL)**

**{**

**delete pNode;**

**pNode = NULL;**

**}**

**}**

**bool BTree::search(Node \*pNode, const int &key) //查找关键字**

**{**

**//Node类型的参数pNode：树结点指针；整型参数key:关键字的值**

**if (pNode==NULL) //检测节点指针是否为空**

**{**

**return false;**

**}**

**else**

**{**

**int i;**

**for (i=0; i<pNode->keyNum && key>\*(pNode->keyValue+i); ++i)//找到使key<=pNode->keyValue[i]成立的最小下标i**

**{**

**}**

**if (i<pNode->keyNum && key==pNode->keyValue[i])**

**{**

**return true;**

**}**

**else**

**{**

**if (pNode->isLeaf) //检查该节点是否为叶子节点**

**{**

**return false;**

**}**

**else**

**{**

**return search(pNode->pChild[i], key);**

**}**

**}**

**}**

**}**

**void BTree::splitChild(Node \*pParent, int nChildIndex, Node \*pChild) //分裂子节点**

**{**

**// Node类型的参数pParent:父节点；pChild：孩子结点；整型参数nChildIndex：孩子下标**

**//将pChild分裂成pRightNode和pChild两个节点**

**Node \*pRightNode = new Node();//分裂后的右节点**

**pRightNode->isLeaf = pChild->isLeaf;**

**pRightNode->keyNum = KEY\_MIN;**

**int i;**

**for (i=0; i<KEY\_MIN; ++i)//拷贝关键字的值**

**{**

**pRightNode->keyValue[i] = pChild->keyValue[i+CHILD\_MIN];**

**}**

**if (!pChild->isLeaf) //如果不是叶子节点，拷贝孩子节点指针**

**{**

**for (i=0; i<CHILD\_MIN; ++i)**

**{**

**pRightNode->pChild[i] = pChild->pChild[i+CHILD\_MIN];**

**}**

**}**

**pChild->keyNum = KEY\_MIN; //更新左子树的关键字个数**

**for (i=pParent->keyNum; i>nChildIndex; --i)//将父节点中的nChildIndex后的所有关键字的值和子树指针向后移一位**

**{**

**pParent->pChild[i+1] = pParent->pChild[i];**

**pParent->keyValue[i] = pParent->keyValue[i-1];**

**}**

**++pParent->keyNum; //更新父节点的关键字个数**

**pParent->pChild[nChildIndex+1] = pRightNode; //存储右子树指针**

**pParent->keyValue[nChildIndex] = pChild->keyValue[KEY\_MIN];//把节点的中间值提到父节点**

**}**

**//在非满节点中插入关键字**

**void BTree::insertNonFull(Node \*pNode, const int &key)**

**{**

**// Node类型的参数pNode：树结点指针；整型参数key:关键字的值**

**int i = pNode->keyNum; //获取节点内关键字个数**

**if (pNode->isLeaf) //pNode是叶子节点**

**{**

**while (i>0&&key<pNode->keyValue[i-1]) //从后往前，查找关键字的插入位置**

**{**

**pNode->keyValue[i] = pNode->keyValue[i-1]; //向后移位**

**--i;**

**}**

**pNode->keyValue[i] = key; //插入关键字的值**

**++pNode->keyNum; //更新节点关键字的个数**

**}**

**else//pNode是内节点**

**{**

**while(i>0&&key<pNode->keyValue[i-1]) //从后往前，查找关键字的插入的子树**

**--i;**

**Node \*pChild = pNode->pChild[i]; //目标子树结点指针**

**if (pChild->keyNum==KEY\_MAX) //子树节点已满**

**{**

**splitChild(pNode, i, pChild);//分裂子树节点**

**if(key>pNode->keyValue[i]) //确定目标子树**

**pChild = pNode->pChild[i+1];**

**}**

**insertNonFull(pChild, key); //插入关键字到目标子树节点**

**}**

**}**

**//打印树**

**void BTree::displayBtree(Node \*pNode)**

**{**

**//Node类型的参数pNode：树结点指针**

**if (pNode!=NULL)**

**{**

**int i;**

**for (i=0; i<pNode->keyNum; ++i)**

**{**

**if (!pNode->isLeaf)**

**{**

**displayBtree(pNode->pChild[i]);**

**}**

**cout<<pNode->keyValue[i]<<" ";**

**}**

**}**

**}**

**//合并两个子节点**

**void BTree::mergeChild(Node \*pParent, int index)**

**{**

**//Node类型的参数pParent：树结点指针；整型参数index:关键字key的下标**

**Node \*pChild1 = pParent->pChild[index]; //节点pNode中前于key的子节点**

**Node \*pChild2 = pParent->pChild[index+1]; //节点pNode中后于key的子节点**

**//将pChild2数据合并到pChild1**

**pChild1->keyNum = KEY\_MAX;**

**pChild1->keyValue[KEY\_MIN] = pParent->keyValue[index];//将父节点index的值下移**

**int i;**

**for (i=0; i<KEY\_MIN; ++i) //将 pChild2添加到pChild1中**

**{**

**pChild1->keyValue[i+KEY\_MIN+1] = pChild2->keyValue[i];**

**}**

**if (!pChild1->isLeaf)**

**{**

**for (i=0; i<CHILD\_MIN; ++i)**

**{**

**pChild1->pChild[i+CHILD\_MIN] = pChild2->pChild[i];**

**}**

**}**

**//父节点删除index的key，index后的往前移一位**

**--pParent->keyNum;**

**for(i=index; i<pParent->keyNum; ++i)**

**{**

**pParent->keyValue[i] = pParent->keyValue[i+1];**

**pParent->pChild[i+1] = pParent->pChild[i+2];**

**}**

**deleteNode(pChild2); //删除pChild2**

**}**

**//递归的删除关键字**

**void BTree::recursive\_delete(Node \*pNode, const int &key)**

**{**

**//Node类型的参数pNode：树结点指针；整型参数key:关键字的值**

**int i=0;**

**while(i<pNode->keyNum&&key>pNode->keyValue[i])**

**++i;**

**if (i<pNode->keyNum&&key==pNode->keyValue[i])//关键字key在节点pNode中**

**{**

**if (pNode->isLeaf)//pNode是个叶节点**

**{**

**//从pNode中删除key**

**--pNode->keyNum;**

**for (; i<pNode->keyNum; ++i)**

**{**

**pNode->keyValue[i] = pNode->keyValue[i+1];**

**}**

**return;**

**}**

**else//pNode是个内节点**

**{**

**Node \*pChildPrev = pNode->pChild[i];//节点pNode中前于key的子节点**

**Node \*pChildNext = pNode->pChild[i+1];//节点pNode中后于key的子节点**

**if (pChildPrev->keyNum>=CHILD\_MIN)//节点pChildPrev中至少包含CHILD\_MIN个关键字**

**{**

**int prevKey = getbefore(pChildPrev); //获取key的前驱关键字**

**recursive\_delete(pChildPrev, prevKey);**

**pNode->keyValue[i] = prevKey; //替换成key的前驱关键字**

**return;**

**}**

**else if (pChildNext->keyNum>=CHILD\_MIN)//节点pChildNext中至少包含CHILD\_MIN个关键字**

**{**

**int nextKey = getafter(pChildNext); //获取key的后继关键字**

**recursive\_delete(pChildNext, nextKey);**

**pNode->keyValue[i] = nextKey; //替换成key的后继关键字**

**return;**

**}**

**else//节点pChildPrev和pChildNext中都只包含CHILD\_MIN-1个关键字**

**{**

**mergeChild(pNode, i);**

**recursive\_delete(pChildPrev, key);**

**}**

**}**

**}**

**else//关键字key不在节点pNode中**

**{**

**Node \*pChildNode = pNode->pChild[i];//包含key的子树根节点**

**if (pChildNode->keyNum==KEY\_MIN)//只有t-1个关键字**

**{**

**Node \*pLeft = i>0 ? pNode->pChild[i-1] : NULL; //左兄弟节点**

**Node \*pRight = i<pNode->keyNum ? pNode->pChild[i+1] : NULL;//右兄弟节点**

**int j;**

**if (pLeft&&pLeft->keyNum>=CHILD\_MIN)//左兄弟节点至少有CHILD\_MIN个关键字**

**{**

**//父节点中i-1的关键字下移至pChildNode中**

**for (j=pChildNode->keyNum; j>0; --j)**

**{**

**pChildNode->keyValue[j] = pChildNode->keyValue[j-1];**

**}**

**pChildNode->keyValue[0] = pNode->keyValue[i-1];**

**if (!pLeft->isLeaf)**

**{**

**for (j=pChildNode->keyNum+1; j>0; --j) //pLeft节点中合适的子女指针移植到pChildNode中**

**{**

**pChildNode->pChild[j] = pChildNode->pChild[j-1];**

**}**

**pChildNode->pChild[0] = pLeft->pChild[pLeft->keyNum];**

**}**

**++pChildNode->keyNum;**

**pNode->keyValue[i] = pLeft->keyValue[pLeft->keyNum-1];//pLeft节点中的最大关键字上升到pNode中**

**--pLeft->keyNum;**

**}**

**else if (pRight&&pRight->keyNum>=CHILD\_MIN)//右兄弟节点至少有CHILD\_MIN个关键字**

**{**

**//父节点中i的关键字下移至pChildNode中**

**pChildNode->keyValue[pChildNode->keyNum] = pNode->keyValue[i];**

**++pChildNode->keyNum;**

**pNode->keyValue[i] = pRight->keyValue[0];//pRight节点中的最小关键字上升到pNode中**

**--pRight->keyNum;**

**for (j=0; j<pRight->keyNum; ++j)**

**{**

**pRight->keyValue[j] = pRight->keyValue[j+1];**

**}**

**if (!pRight->isLeaf)**

**{**

**pChildNode->pChild[pChildNode->keyNum] = pRight->pChild[0];//pRight节点中合适的子女指针移植到pChildNode中**

**for (j=0; j<=pRight->keyNum; ++j)**

**{**

**pRight->pChild[j] = pRight->pChild[j+1];**

**}**

**}**

**}**

**//左右兄弟节点都只包含CHILD\_MIN-1个节点**

**else if (pLeft)//与左兄弟合并**

**{**

**mergeChild(pNode, i-1);**

**pChildNode = pLeft;**

**}**

**else if (pRight)//与右兄弟合并**

**{**

**mergeChild(pNode, i);**

**pChildNode = pRight;**

**}**

**}**

**recursive\_delete(pChildNode, key);**

**}**

**}**

**int BTree::getbefore(Node \*pNode)//找到前驱关键字**

**{**

**//Node类型的参数pNode：树结点指针**

**while (!pNode->isLeaf)**

**{**

**pNode = pNode->pChild[pNode->keyNum];**

**}**

**return pNode->keyValue[pNode->keyNum-1];**

**}**

**int BTree::getafter(Node \*pNode)//找到后继关键字**

**{**

**//Node类型的参数pNode：树结点指针**

**while (!pNode->isLeaf)**

**{**

**pNode = pNode->pChild[0];**

**}**

**return pNode->keyValue[0];**

**}**

**int main()**

**{**

**BTree cb;**

**int n,key;**

**bool visit;**

**char ch='y';**

**do{**

**cout<<"\*\* B\_Tree的实现及分析 \*\*"<<endl;**

**cout<<"\*\* 1.插入数据 \*\*"<<endl;**

**cout<<"\*\* 2.删除数据 \*\*"<<endl;**

**cout<<"\*\* 3.查找数据 \*\*"<<endl;**

**cout<<"\*\* 4.输出B\_Tree \*\*"<<endl;**

**cout<<"\*\* 5.退出系统 \*\*"<<endl;**

**cout<<"请选择:"<<endl;**

**cin>>n;**

**visit=false;**

**switch(n)**

**{**

**case 1:**

**cout<<"请输入待插入数据:"<<endl;**

**cin>>key;**

**visit=cb.insert(key);**

**if(visit)**

**cout<<"插入成功!"<<endl;**

**else**

**cout<<"插入失败!"<<endl;**

**break;**

**case 2:**

**cout<<"请输入想要删除的数据:"<<endl;**

**cin>>key;**

**visit=cb.deletekey(key);**

**if(visit)**

**cout<<"删除成功!"<<endl;**

**else**

**cout<<"删除失败!"<<endl;**

**break;**

**case 3:**

**cout<<"请输入待查找的数据:"<<endl;**

**cin>>key;**

**visit=cb.search(cb.GetRoot(),key);**

**if(visit)**

**cout<<"查找成功!"<<endl;**

**else**

**cout<<"查找失败!"<<endl;**

**break;**

**case 4:**

**cb.display();**

**break;**

**case 5:**

**cout<<"程序运行结束!"<<endl;**

**ch='n';**

**break;**

**default:**

**cout<<"输入错误!!"<<endl;**

**break;**

**}**

**system("pause");**

**system("cls");**

**}while(ch=='y'||ch=='Y');**

**return 0;**

**}**