# 机房预约系统

## 1、机房预约系统需求

### 1.1 系统简介

* 学校现有几个规格不同的机房，由于使用时经常出现"撞车"现象,现开发一套机房预约系统，解决这一问题。



### 1.2 身份简介

分别有三种身份使用该程序

* **学生代表**：申请使用机房
* **教师**：审核学生的预约申请
* **管理员**：给学生、教师创建账号

### 1.3 机房简介

机房总共有3间

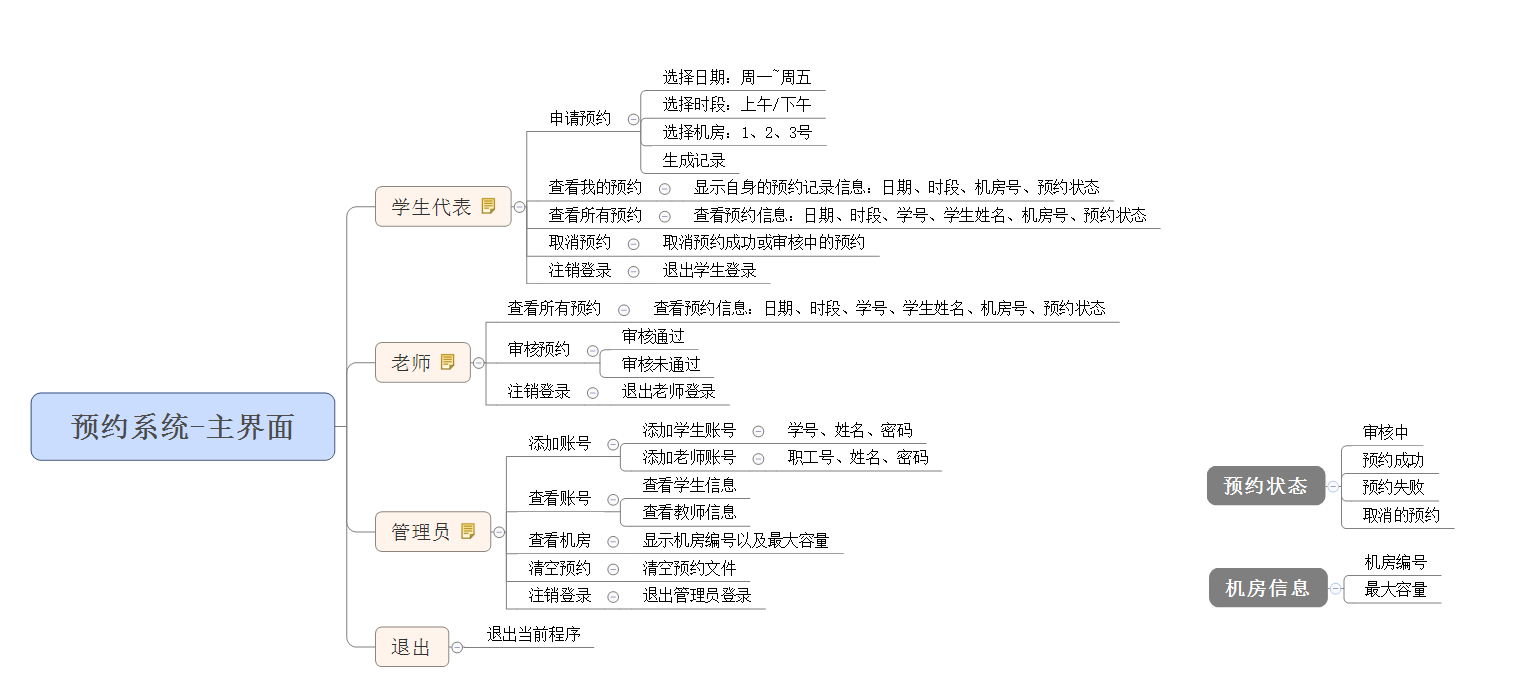
* 1号机房 --- 最大容量20人
* 2号机房 --- 最多容量50人
* 3号机房 --- 最多容量100人

### 1.4 申请简介

* 申请的订单每周由管理员负责清空。
* 学生可以预约未来一周内的机房使用，预约的日期为周一至周五，预约时需要选择预约时段（上午、下午）
* 教师来审核预约，依据实际情况审核预约通过或者不通过

### 1.5 系统具体需求

* 首先进入登录界面，可选登录身份有：
  + 学生代表
  + 老师
  + 管理员
  + 退出
* 每个身份都需要进行验证后，进入子菜单
  + 学生需要输入 ：学号、姓名、登录密码
  + 老师需要输入：职工号、姓名、登录密码
  + 管理员需要输入：管理员姓名、登录密码
* 学生具体功能
  + 申请预约 --- 预约机房
  + 查看自身的预约 --- 查看自己的预约状态
  + 查看所有预约 --- 查看全部预约信息以及预约状态
  + 取消预约 --- 取消自身的预约，预约成功或审核中的预约均可取消
  + 注销登录 --- 退出登录
* 教师具体功能
  + 查看所有预约 --- 查看全部预约信息以及预约状态
  + 审核预约 --- 对学生的预约进行审核
  + 注销登录 --- 退出登录
* 管理员具体功能
  + 添加账号 --- 添加学生或教师的账号，需要检测学生编号或教师职工号是否重复
  + 查看账号 --- 可以选择查看学生或教师的全部信息
  + 查看机房 --- 查看所有机房的信息
  + 清空预约 --- 清空所有预约记录
  + 注销登录 --- 退出登录



## 2、创建项目

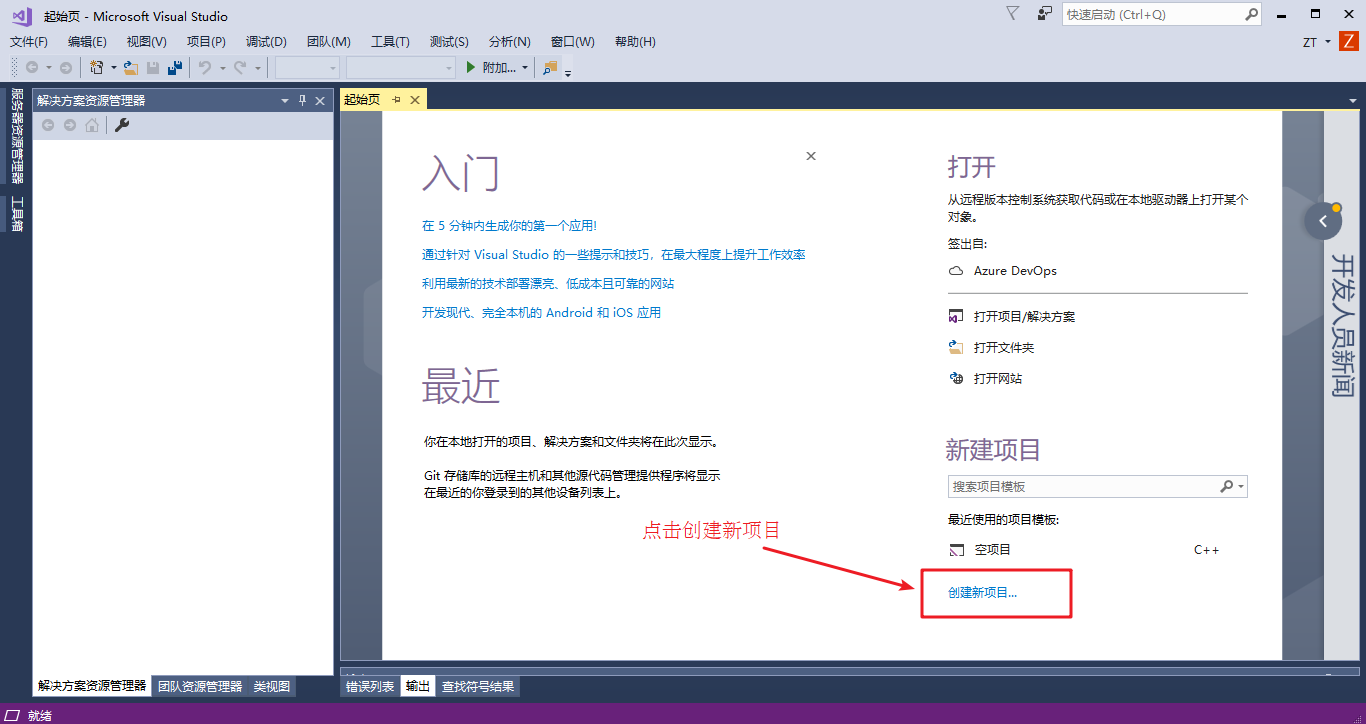
创建项目步骤如下：

* 创建新项目
* 添加文件

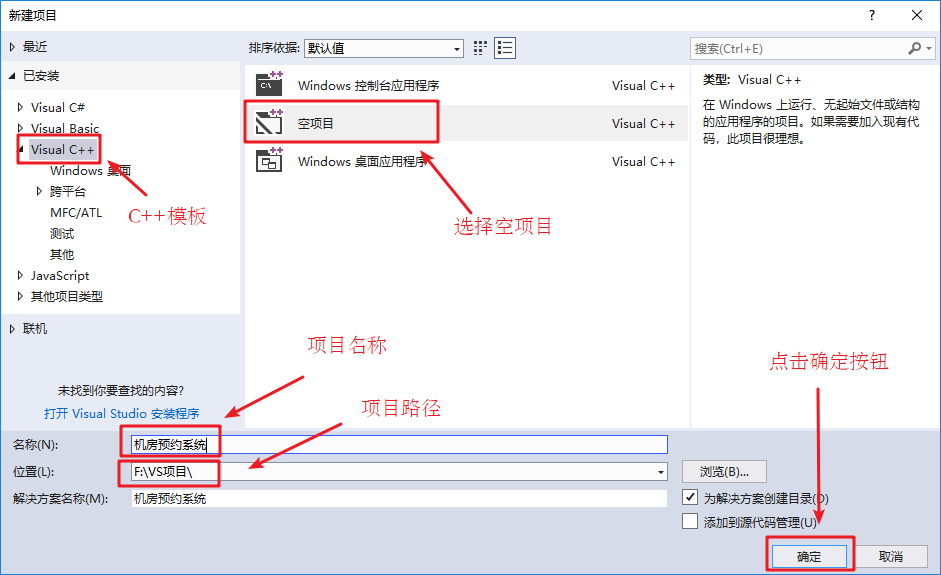
### 2.1 创建项目

* 打开vs2017后，点击创建新项目，创建新的C++项目

如图：

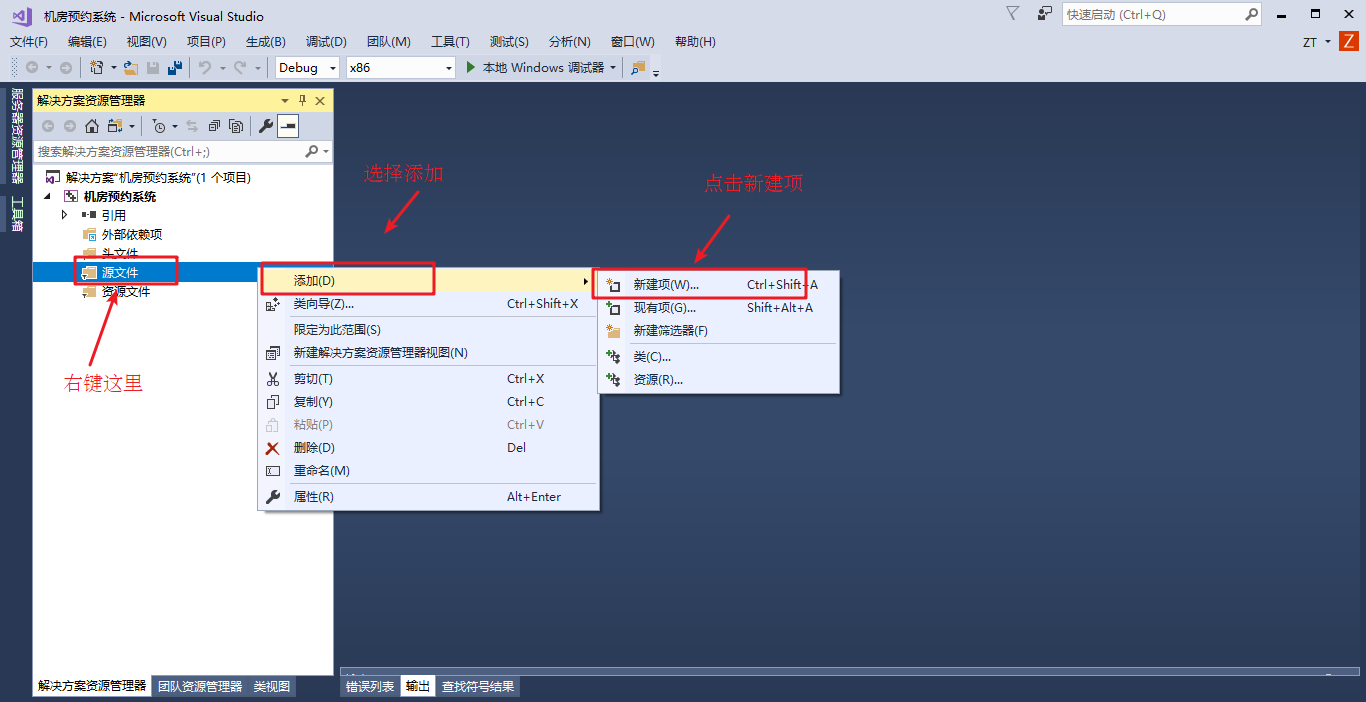


* 填写项目名称以及选取项目路径，点击确定生成项目

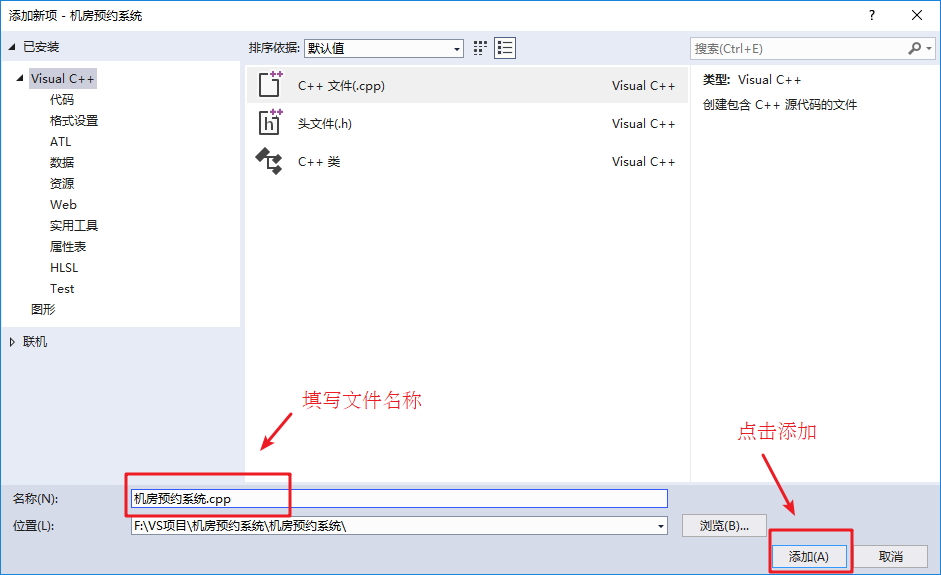


### 2.2 添加文件

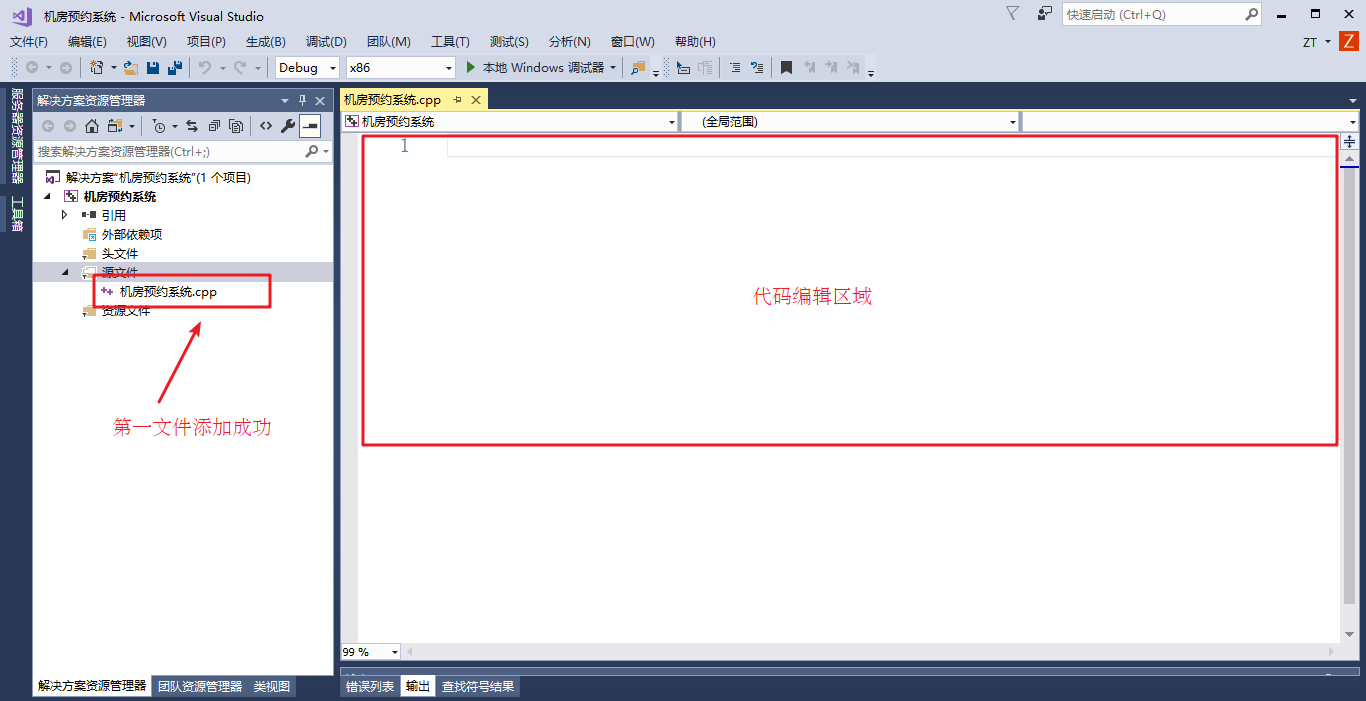
* 右键源文件，进行添加文件操作



* 填写文件名称，点击添加



* 生成文件成功，效果如下图



## 3、创建主菜单

**功能描述：**

* 设计主菜单，与用户进行交互

### 3.1 菜单实现

* 在主函数main中添加菜单提示，代码如下：

int main() {  
  
 cout << "====================== 欢迎来到传智播客机房预约系统 ====================="   
 << endl;  
 cout << endl << "请输入您的身份" << endl;  
 cout << "\t\t -------------------------------\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 1.学生代表 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 2.老 师 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 3.管 理 员 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 0.退 出 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t -------------------------------\n";  
 cout << "输入您的选择: ";  
  
 system("pause");  
  
 return 0;  
}

运行效果如图：



### 3.2 搭建接口

* 接受用户的选择，搭建接口
* 在main中添加代码

int main() {  
  
 int select = 0;  
  
 while (true)  
 {  
  
 cout << "====================== 欢迎来到传智播客机房预约系统 =====================" << endl;  
 cout << endl << "请输入您的身份" << endl;  
 cout << "\t\t -------------------------------\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 1.学生代表 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 2.老 师 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 3.管 理 员 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 0.退 出 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t -------------------------------\n";  
 cout << "输入您的选择: ";  
  
 cin >> select; //接受用户选择  
  
 switch (select)  
 {  
 case 1: //学生身份  
 break;  
 case 2: //老师身份  
 break;  
 case 3: //管理员身份  
 break;  
 case 0: //退出系统  
 break;  
 default:  
 cout << "输入有误，请重新选择！" << endl;  
 system("pause");  
 system("cls");  
 break;  
 }  
  
 }  
 system("pause");  
 return 0;  
}

测试，输入0、1、2、3会重新回到界面，输入其他提示输入有误，清屏后重新选择

效果如图：



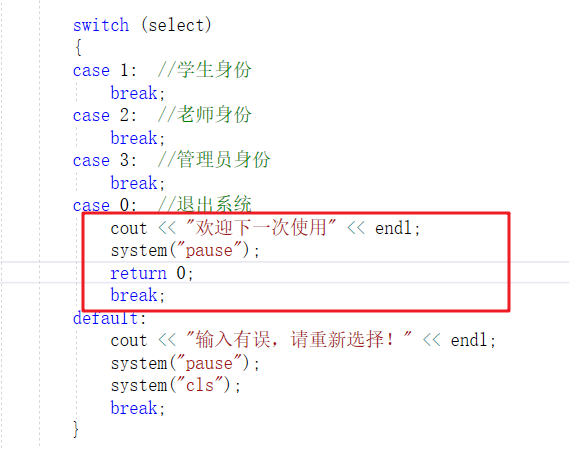
至此，界面搭建完毕

## 4、 退出功能实现

### 4.1 退出功能实现

在main函数分支 0 选项中，添加退出程序的代码：

cout << "欢迎下一次使用"<<endl;  
 system("pause");  
 return 0;



### 4.2 测试退出功能

运行程序，效果如图：



至此，退出程序功能实现

## 5、 创建身份类

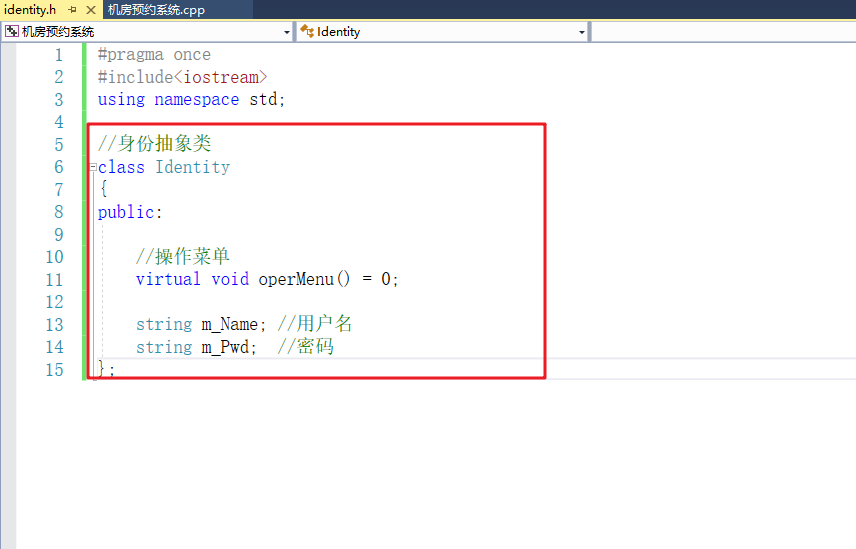
### 5.1 身份的基类

* 在整个系统中，有三种身份，分别为：学生代表、老师以及管理员
* 三种身份有其共性也有其特性，因此我们可以将三种身份抽象出一个身份基类**identity**
* 在头文件下创建Identity.h文件

Identity.h中添加如下代码：

#pragma once  
#include<iostream>  
using namespace std;  
  
//身份抽象类  
class Identity  
{  
public:  
  
 //操作菜单  
 virtual void operMenu() = 0;  
  
 string m\_Name; //用户名  
 string m\_Pwd; //密码  
};

效果如图：



### 5.2 学生类

#### 5.2.1 功能分析

* 学生类主要功能是可以通过类中成员函数，实现预约实验室操作
* 学生类中主要功能有：
  + 显示学生操作的菜单界面
  + 申请预约
  + 查看自身预约
  + 查看所有预约
  + 取消预约

#### 5.2.2 类的创建

* 在头文件以及源文件下创建 student.h 和 student.cpp文件

student.h中添加如下代码：

#pragma once  
#include<iostream>  
using namespace std;  
#include "identity.h"  
  
//学生类  
class Student :public Identity  
{  
public:  
 //默认构造  
 Student();  
  
 //有参构造(学号、姓名、密码)  
 Student(int id, string name, string pwd);  
  
 //菜单界面  
 virtual void operMenu();   
  
 //申请预约  
 void applyOrder();   
  
 //查看我的预约  
 void showMyOrder();   
  
 //查看所有预约  
 void showAllOrder();   
  
 //取消预约  
 void cancelOrder();  
   
 //学生学号  
 int m\_Id;  
  
};

student.cpp中添加如下代码：

#include "student.h"  
  
//默认构造  
Student::Student()  
{  
}  
  
//有参构造(学号、姓名、密码)  
Student::Student(int id, string name, string pwd)  
{  
}  
  
//菜单界面  
void Student::operMenu()  
{  
}  
  
//申请预约  
void Student::applyOrder()  
{  
  
}  
  
//查看我的预约  
void Student::showMyOrder()  
{  
  
}  
  
//查看所有预约  
void Student::showAllOrder()  
{  
  
}  
  
//取消预约  
void Student::cancelOrder()  
{  
  
}

### 5.3 老师类

#### 5.3.1 功能分析

* 教师类主要功能是查看学生的预约，并进行审核
* 教师类中主要功能有：
  + 显示教师操作的菜单界面
  + 查看所有预约
  + 审核预约

#### 5.3.2 类的创建

* 在头文件以及源文件下创建 teacher.h 和 teacher.cpp文件

teacher.h中添加如下代码：

#pragma once  
#define \_CRT\_SECURE\_NO\_WARNINGS  
#include<iostream>  
using namespace std;  
#include "identity.h"  
  
class Teacher :public Identity  
{  
public:  
  
 //默认构造  
 Teacher();  
  
 //有参构造 (职工编号，姓名，密码)  
 Teacher(int empId, string name, string pwd);  
  
 //菜单界面  
 virtual void operMenu();  
  
 //查看所有预约  
 void showAllOrder();   
  
 //审核预约  
 void validOrder();   
  
 int m\_EmpId; //教师编号  
  
};

* teacher.cpp中添加如下代码:

#include"teacher.h"  
  
//默认构造  
Teacher::Teacher()  
{  
}  
  
//有参构造 (职工编号，姓名，密码)  
Teacher::Teacher(int empId, string name, string pwd)  
{  
}  
  
//菜单界面  
void Teacher::operMenu()  
{  
}  
  
//查看所有预约  
void Teacher::showAllOrder()  
{  
}  
  
//审核预约  
void Teacher::validOrder()  
{  
}

### 5.4 管理员类

#### 5.4.1 功能分析

* 管理员类主要功能是对学生和老师账户进行管理，查看机房信息以及清空预约记录
* 管理员类中主要功能有：
  + 显示管理员操作的菜单界面
  + 添加账号
  + 查看账号
  + 查看机房信息
  + 清空预约记录

#### 5.4.2 类的创建

* 在头文件以及源文件下创建 manager.h 和 manager.cpp文件

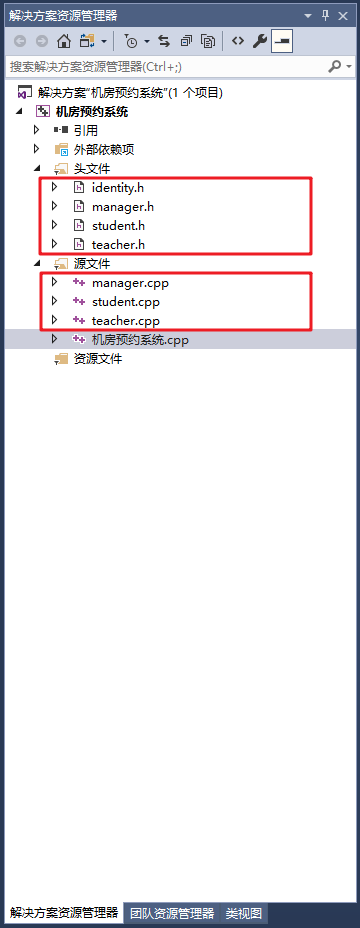
manager.h中添加如下代码：

#pragma once  
#include<iostream>  
using namespace std;  
#include "identity.h"  
  
class Manager :public Identity  
{  
public:  
  
 //默认构造  
 Manager();  
  
 //有参构造 管理员姓名，密码  
 Manager(string name, string pwd);  
  
 //选择菜单  
 virtual void operMenu();  
  
 //添加账号   
 void addPerson();  
  
 //查看账号  
 void showPerson();  
  
 //查看机房信息  
 void showComputer();  
  
 //清空预约记录  
 void cleanFile();  
  
};

* manager.cpp中添加如下代码:

#include "manager.h"  
  
//默认构造  
Manager::Manager()  
{  
}  
  
//有参构造  
Manager::Manager(string name, string pwd)  
{  
}  
  
//选择菜单  
void Manager::operMenu()  
{  
}  
  
//添加账号   
void Manager::addPerson()  
{  
}  
  
//查看账号  
void Manager::showPerson()  
{  
}  
  
//查看机房信息  
void Manager::showComputer()  
{  
}  
  
//清空预约记录  
void Manager::cleanFile()  
{  
}

至此，所有身份类创建完毕，效果如图：



## 6、 登录模块

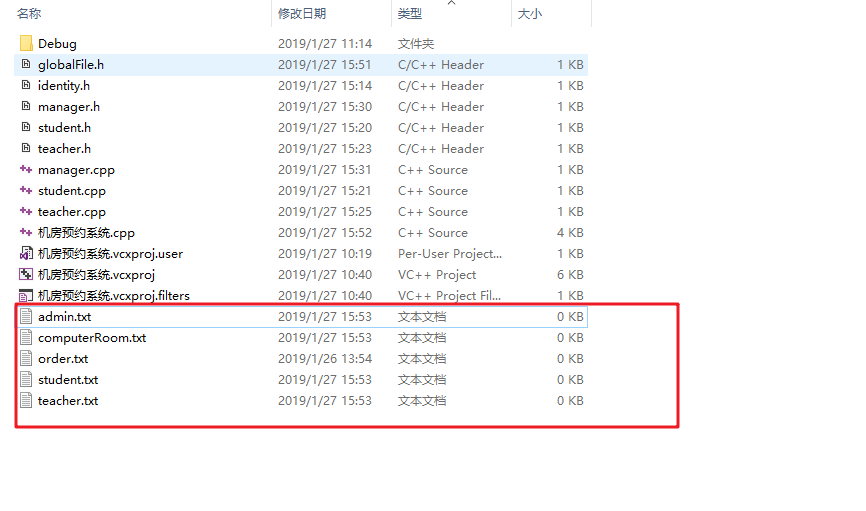
### 6.1 全局文件添加

功能描述：

* 不同的身份可能会用到不同的文件操作，我们可以将所有的文件名定义到一个全局的文件中
* 在头文件中添加 **globalFile.h** 文件
* 并添加如下代码：

#pragma once   
  
//管理员文件  
#define ADMIN\_FILE "admin.txt"  
//学生文件  
#define STUDENT\_FILE "student.txt"  
//教师文件  
#define TEACHER\_FILE "teacher.txt"  
//机房信息文件  
#define COMPUTER\_FILE "computerRoom.txt"  
//订单文件  
#define ORDER\_FILE "order.txt"

并且在同级目录下，创建这几个文件



### 6.2 登录函数封装

功能描述：

* 根据用户的选择，进入不同的身份登录

在预约系统的.cpp文件中添加全局函数 void LoginIn(string fileName, int type)

参数：

* fileName --- 操作的文件名
* type --- 登录的身份 （1代表学生、2代表老师、3代表管理员）

LoginIn中添加如下代码：

#include "globalFile.h"  
#include "identity.h"  
#include <fstream>  
#include <string>  
  
  
//登录功能  
void LoginIn(string fileName, int type)  
{  
  
 Identity \* person = NULL;  
  
 ifstream ifs;  
 ifs.open(fileName, ios::in);  
  
 //文件不存在情况  
 if (!ifs.is\_open())  
 {  
 cout << "文件不存在" << endl;  
 ifs.close();  
 return;  
 }  
  
 int id = 0;  
 string name;  
 string pwd;  
  
 if (type == 1) //学生登录  
 {  
 cout << "请输入你的学号" << endl;  
 cin >> id;  
 }  
 else if (type == 2) //教师登录  
 {  
 cout << "请输入你的职工号" << endl;  
 cin >> id;  
 }  
  
 cout << "请输入用户名：" << endl;  
 cin >> name;  
  
 cout << "请输入密码： " << endl;  
 cin >> pwd;  
  
  
 if (type == 1)  
 {  
 //学生登录验证  
 }  
 else if (type == 2)  
 {  
 //教师登录验证  
 }  
 else if(type == 3)  
 {  
 //管理员登录验证  
 }  
   
 cout << "验证登录失败!" << endl;  
  
 system("pause");  
 system("cls");  
 return;  
}

* 在main函数的不同分支中，填入不同的登录接口



### 6.3 学生登录实现

在student.txt文件中添加两条学生信息，用于测试

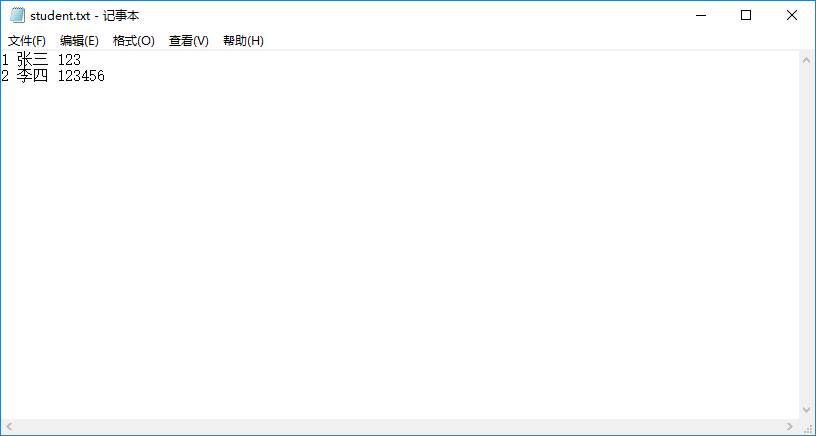
添加信息:

1 张三 123  
2 李四 123456

其中：

* 第一列 代表 **学号**
* 第二列 代表 **学生姓名**
* 第三列 代表 **密码**

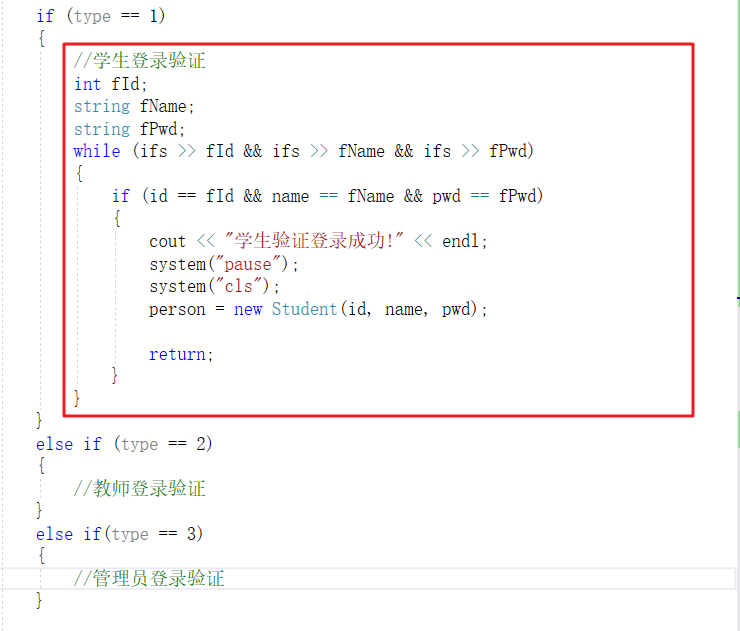
效果图：



在Login函数的学生分支中加入如下代码，验证学生身份

//学生登录验证  
 int fId;  
 string fName;  
 string fPwd;  
 while (ifs >> fId && ifs >> fName && ifs >> fPwd)  
 {  
 if (id == fId && name == fName && pwd == fPwd)  
 {  
 cout << "学生验证登录成功!" << endl;  
 system("pause");  
 system("cls");  
 person = new Student(id, name, pwd);  
   
 return;  
 }  
 }

添加代码效果图



测试：



### 6.4 教师登录实现

在teacher.txt文件中添加一条老师信息，用于测试

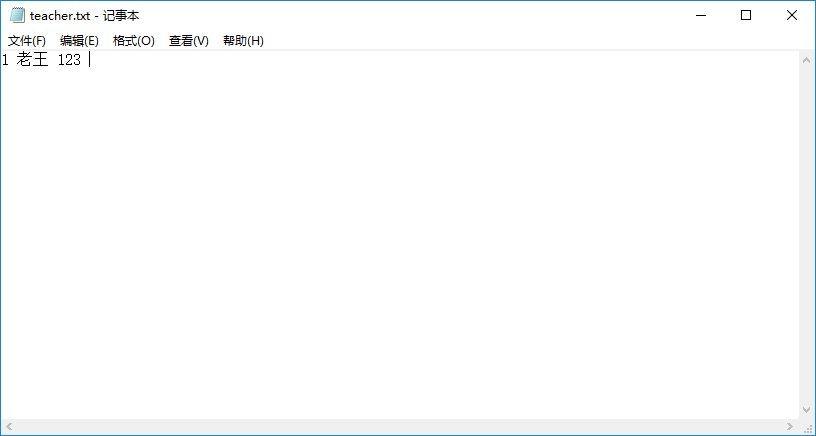
添加信息:

1 老王 123

其中：

* 第一列 代表 **教师职工编号**
* 第二列 代表 **教师姓名**
* 第三列 代表 **密码**

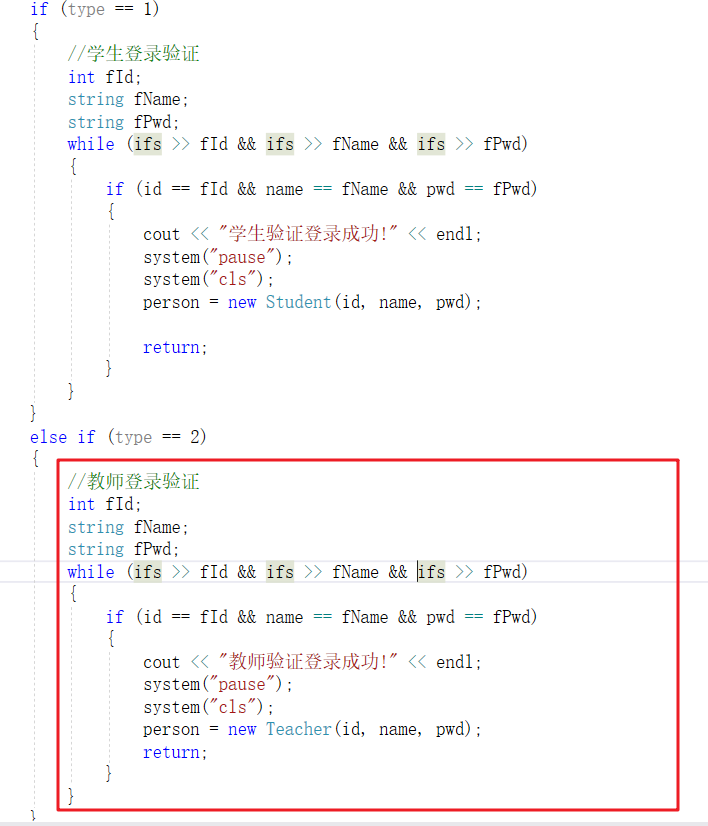
效果图：



在Login函数的教师分支中加入如下代码，验证教师身份

//教师登录验证  
 int fId;  
 string fName;  
 string fPwd;  
 while (ifs >> fId && ifs >> fName && ifs >> fPwd)  
 {  
 if (id == fId && name == fName && pwd == fPwd)  
 {  
 cout << "教师验证登录成功!" << endl;  
 system("pause");  
 system("cls");  
 person = new Teacher(id, name, pwd);  
 return;  
 }  
 }

添加代码效果图



测试：



### 6.5 管理员登录实现

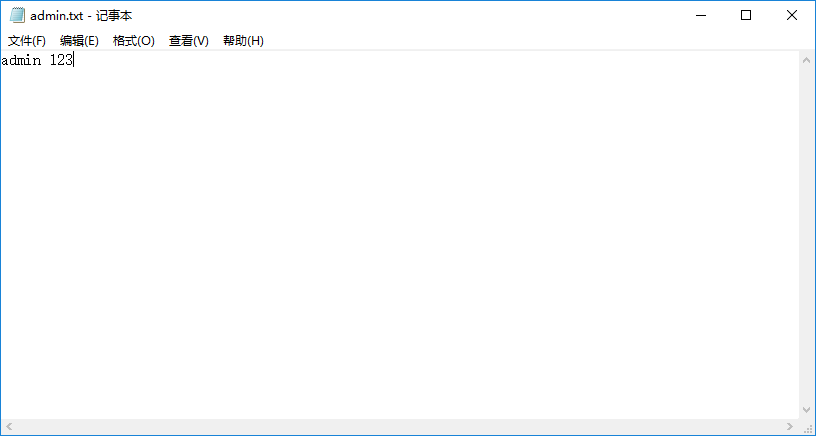
在admin.txt文件中添加一条管理员信息，由于我们只有一条管理员，因此本案例中没有添加管理员的功能

添加信息:

admin 123

其中：admin代表管理员用户名，123代表管理员密码

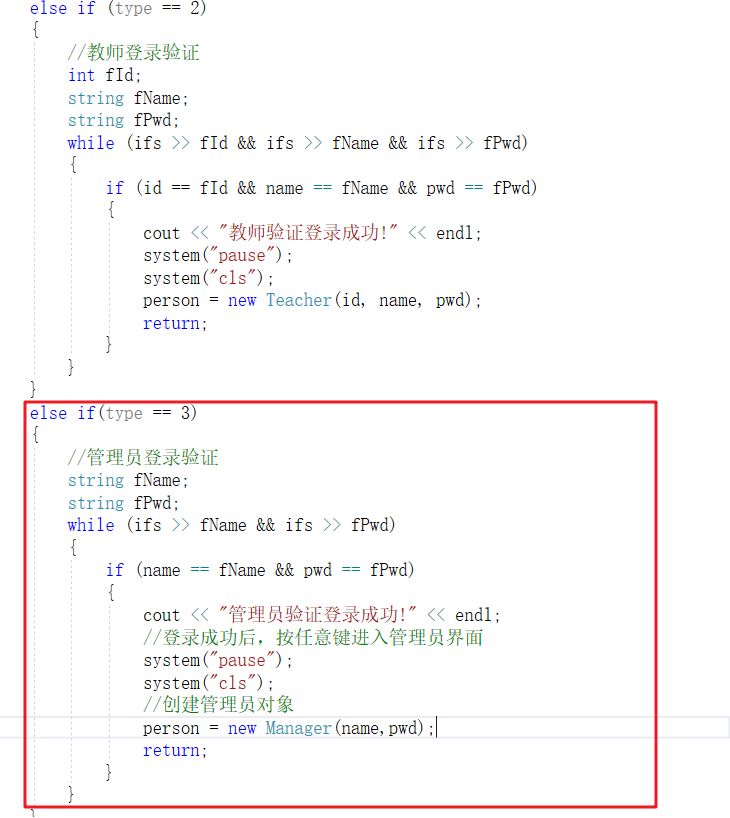
效果图：



在Login函数的管理员分支中加入如下代码，验证管理员身份

//管理员登录验证  
 string fName;  
 string fPwd;  
 while (ifs >> fName && ifs >> fPwd)  
 {  
 if (name == fName && pwd == fPwd)  
 {  
 cout << "验证登录成功!" << endl;  
 //登录成功后，按任意键进入管理员界面  
 system("pause");  
 system("cls");  
 //创建管理员对象  
 person = new Manager(name,pwd);  
 return;  
 }  
 }

添加效果如图：



测试效果如图：



至此，所有身份的登录功能全部实现！

## 7、 管理员模块

### 7.1 管理员登录和注销

#### 7.1.1 构造函数

* 在Manager类的构造函数中，初始化管理员信息，代码如下：

//有参构造  
Manager::Manager(string name, string pwd)  
{  
 this->m\_Name = name;  
 this->m\_Pwd = pwd;  
}

#### 7.1.2 管理员子菜单

* 在机房预约系统.cpp中，当用户登录的是管理员，添加管理员菜单接口
* 将不同的分支提供出来
  + 添加账号
  + 查看账号
  + 查看机房
  + 清空预约
  + 注销登录
* 实现注销功能

添加全局函数 void managerMenu(Identity \* &manager)，代码如下：

//管理员菜单  
void managerMenu(Identity \* &manager)  
{  
 while (true)  
 {  
 //管理员菜单  
 manager->operMenu();  
  
 Manager\* man = (Manager\*)manager;  
 int select = 0;  
  
 cin >> select;  
   
 if (select == 1) //添加账号  
 {  
 cout << "添加账号" << endl;  
 man->addPerson();  
 }  
 else if (select == 2) //查看账号  
 {  
 cout << "查看账号" << endl;  
 man->showPerson();   
 }  
 else if (select == 3) //查看机房  
 {  
 cout << "查看机房" << endl;  
 man->showComputer();  
 }  
 else if (select == 4) //清空预约  
 {  
 cout << "清空预约" << endl;  
 man->cleanFile();  
 }  
 else  
 {  
 delete manager;  
 cout << "注销成功" << endl;  
 system("pause");  
 system("cls");  
 return;  
 }  
 }  
}

#### 7.1.3 菜单功能实现

* 在实现成员函数void Manager::operMenu() 代码如下：

//选择菜单  
void Manager::operMenu()  
{  
 cout << "欢迎管理员："<<this->m\_Name << "登录！" << endl;  
 cout << "\t\t ---------------------------------\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 1.添加账号 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 2.查看账号 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 3.查看机房 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 4.清空预约 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 0.注销登录 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t ---------------------------------\n";  
 cout << "请选择您的操作： " << endl;  
}

#### 7.1.4 接口对接

* 管理员成功登录后，调用管理员子菜单界面
* 在管理员登录验证分支中，添加代码：

//进入管理员子菜单  
 managerMenu(person);

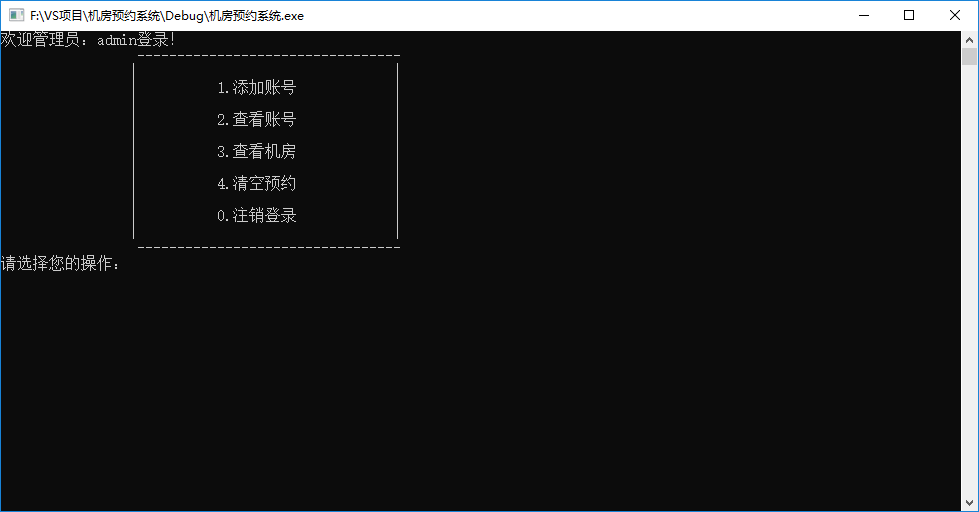
添加效果如：



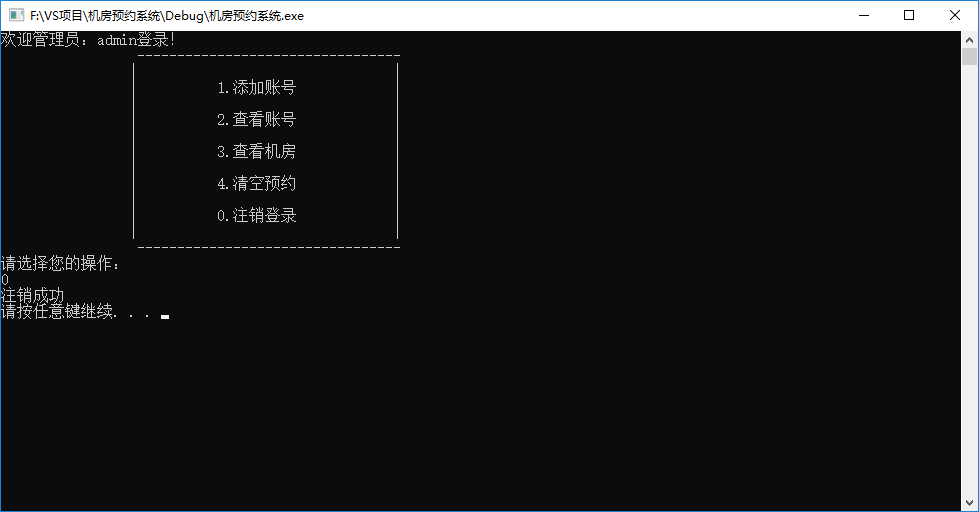
测试对接，效果如图：



登录成功



注销登录：



至此，管理员身份可以成功登录以及注销

### 7.2 添加账号

功能描述：

* 给学生或教师添加新的账号

功能要求：

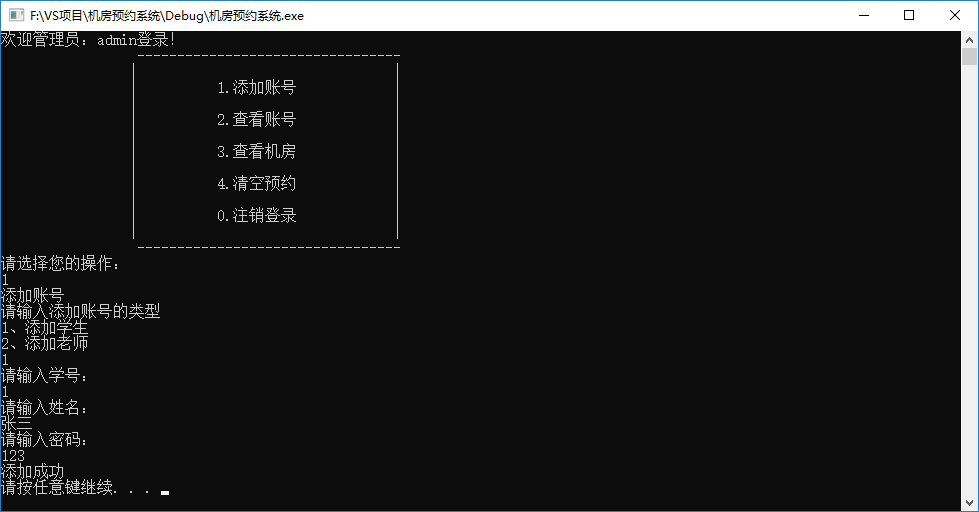
* 添加时学生学号不能重复、教师职工号不能重复

#### 7.2.1 添加功能实现

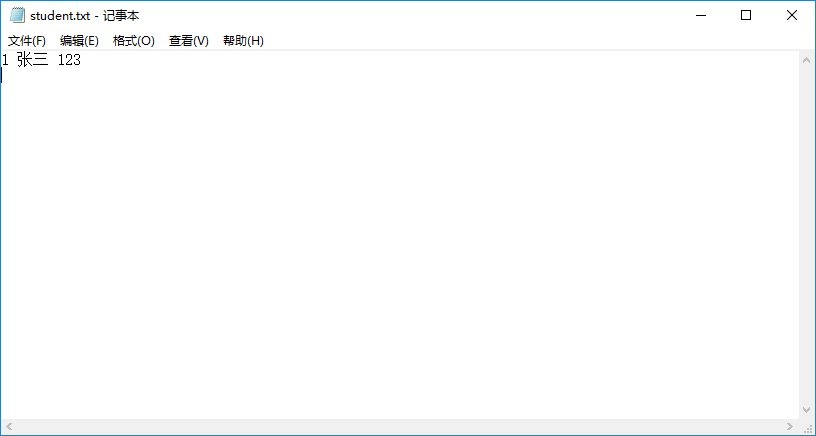
在Manager的**addPerson**成员函数中，实现添加新账号功能，代码如下：

//添加账号   
void Manager::addPerson()  
{  
  
 cout << "请输入添加账号的类型" << endl;  
 cout << "1、添加学生" << endl;  
 cout << "2、添加老师" << endl;  
  
 string fileName;  
 string tip;  
 ofstream ofs;  
  
 int select = 0;  
 cin >> select;  
  
 if (select == 1)  
 {  
 fileName = STUDENT\_FILE;  
 tip = "请输入学号： ";  
 }  
 else  
 {  
 fileName = TEACHER\_FILE;  
 tip = "请输入职工编号：";  
 }  
  
 ofs.open(fileName, ios::out | ios::app);  
 int id;  
 string name;  
 string pwd;  
 cout <<tip << endl;  
 cin >> id;  
  
 cout << "请输入姓名： " << endl;  
 cin >> name;  
  
 cout << "请输入密码： " << endl;  
 cin >> pwd;  
  
 ofs << id << " " << name << " " << pwd << " " << endl;  
 cout << "添加成功" << endl;  
  
 system("pause");  
 system("cls");  
  
 ofs.close();  
}

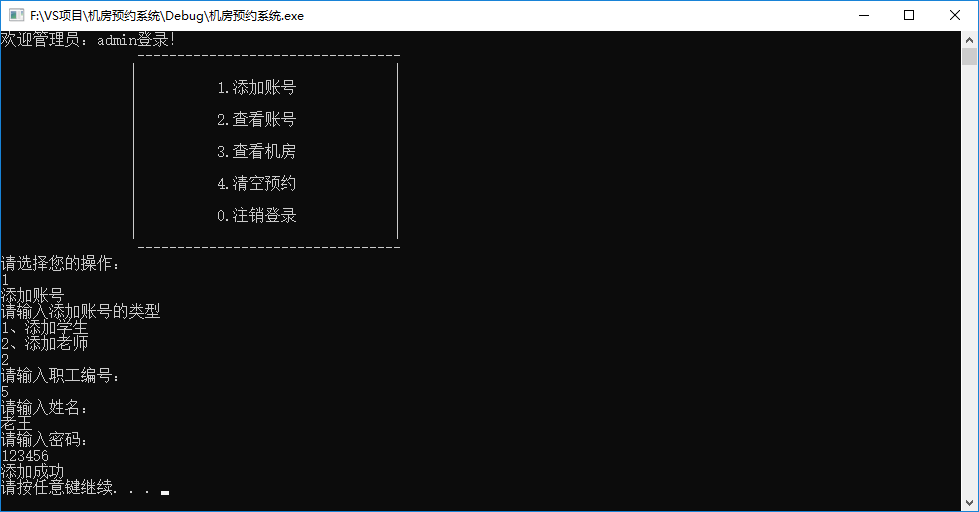
测试添加学生：



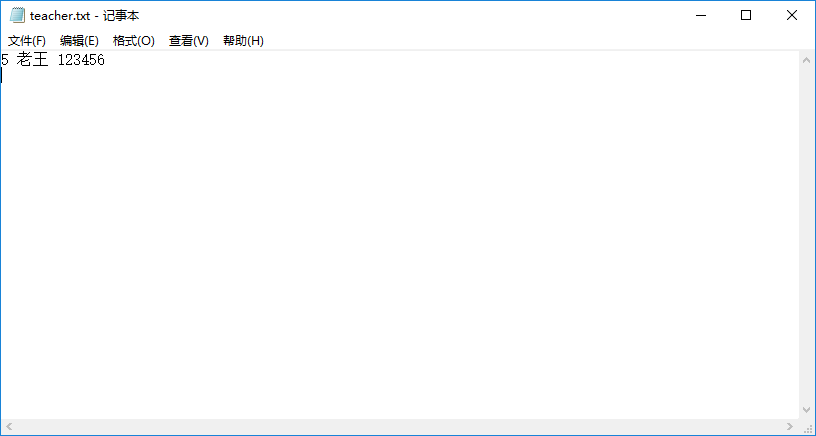
成功在学生文件中添加了一条信息



测试添加教师：



成功在教师文件中添加了一条信息



#### 7.2.2 去重操作

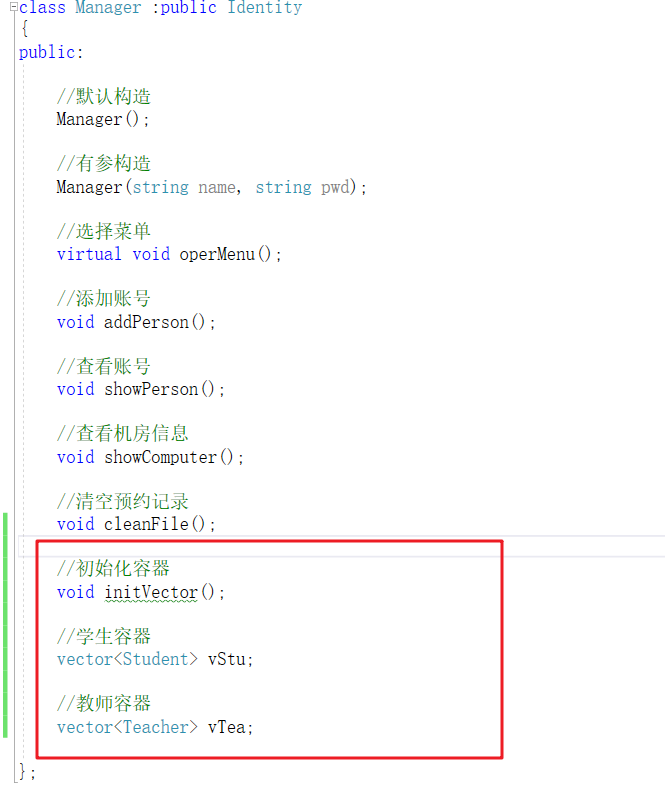
功能描述：添加新账号时，如果是重复的学生编号，或是重复的教师职工编号，提示有误

##### 7.2.2.1 读取信息

* 要去除重复的账号，首先要先将学生和教师的账号信息获取到程序中，方可检测
* 在manager.h中，添加两个容器，用于存放学生和教师的信息
* 添加一个新的成员函数 void initVector() 初始化容器

//初始化容器  
 void initVector();  
  
 //学生容器  
 vector<Student> vStu;  
  
 //教师容器  
 vector<Teacher> vTea;

添加位置如图：



在Manager的有参构造函数中，获取目前的学生和教师信息

代码如下：

void Manager::initVector()  
{  
 //读取学生文件中信息  
 ifstream ifs;  
 ifs.open(STUDENT\_FILE, ios::in);  
 if (!ifs.is\_open())  
 {  
 cout << "文件读取失败" << endl;  
 return;  
 }  
   
 vStu.clear();  
 vTea.clear();  
   
 Student s;  
 while (ifs >> s.m\_Id && ifs >> s.m\_Name && ifs >> s.m\_Pwd)  
 {  
 vStu.push\_back(s);  
 }  
 cout << "当前学生数量为： " << vStu.size() << endl;  
 ifs.close(); //学生初始化  
  
 //读取老师文件信息  
 ifs.open(TEACHER\_FILE, ios::in);  
  
 Teacher t;  
 while (ifs >> t.m\_EmpId && ifs >> t.m\_Name && ifs >> t.m\_Pwd)  
 {  
 vTea.push\_back(t);  
 }  
 cout << "当前教师数量为： " << vTea.size() << endl;  
  
 ifs.close();  
}

在有参构造函数中，调用初始化容器函数

//有参构造  
Manager::Manager(string name, string pwd)  
{  
 this->m\_Name = name;  
 this->m\_Pwd = pwd;  
   
 //初始化容器  
 this->initVector();  
}

测试，运行代码可以看到测试代码获取当前学生和教师数量



##### 7.2.2.2 去重函数封装

在manager.h文件中添加成员函数bool checkRepeat(int id, int type);

//检测重复 参数:(传入id，传入类型) 返回值：(true 代表有重复，false代表没有重复)  
 bool checkRepeat(int id, int type);

在manager.cpp文件中实现成员函数 bool checkRepeat(int id, int type);

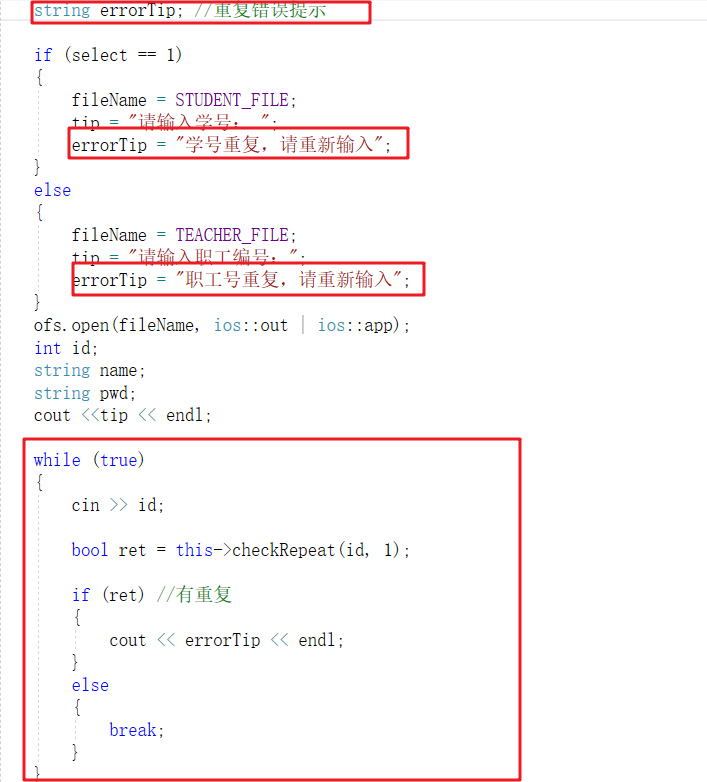
bool Manager::checkRepeat(int id, int type)  
{  
 if (type == 1)  
 {  
 for (vector<Student>::iterator it = vStu.begin(); it != vStu.end(); it++)  
 {  
 if (id == it->m\_Id)  
 {  
 return true;  
 }  
 }  
 }  
 else  
 {  
 for (vector<Teacher>::iterator it = vTea.begin(); it != vTea.end(); it++)  
 {  
 if (id == it->m\_EmpId)  
 {  
 return true;  
 }  
 }  
 }  
 return false;  
}

##### 7.2.2.3 添加去重操作

在添加学生编号或者教师职工号时，检测是否有重复，代码如下：

string errorTip; //重复错误提示  
  
 if (select == 1)  
 {  
 fileName = STUDENT\_FILE;  
 tip = "请输入学号： ";  
 errorTip = "学号重复，请重新输入";  
 }  
 else  
 {  
 fileName = TEACHER\_FILE;  
 tip = "请输入职工编号：";  
 errorTip = "职工号重复，请重新输入";  
 }  
 ofs.open(fileName, ios::out | ios::app);  
 int id;  
 string name;  
 string pwd;  
 cout <<tip << endl;  
  
 while (true)  
 {  
 cin >> id;  
  
 bool ret = this->checkRepeat(id, 1);  
  
 if (ret) //有重复  
 {  
 cout << errorTip << endl;  
 }  
 else  
 {  
 break;  
 }  
 }

代码位置如图：



检测效果：



##### 7.2.2.4 bug解决

bug描述：

* 虽然可以检测重复的账号，但是刚添加的账号由于没有更新到容器中，因此不会做检测
* 导致刚加入的账号的学生号或者职工编号，再次添加时依然可以重复

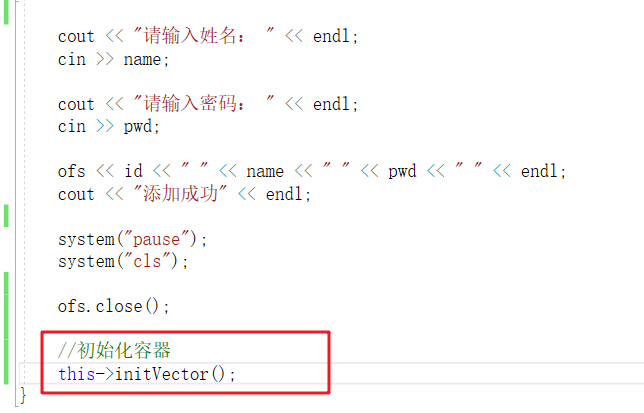
解决方案：

* 在每次添加新账号时，重新初始化容器

在添加完毕后，加入代码：

//初始化容器  
 this->initVector();

位置如图：



再次测试，刚加入的账号不会重复添加了！

### 7.3 显示账号

功能描述：显示学生信息或教师信息

#### 7.3.1 显示功能实现

在Manager的**showPerson**成员函数中，实现显示账号功能，代码如下：

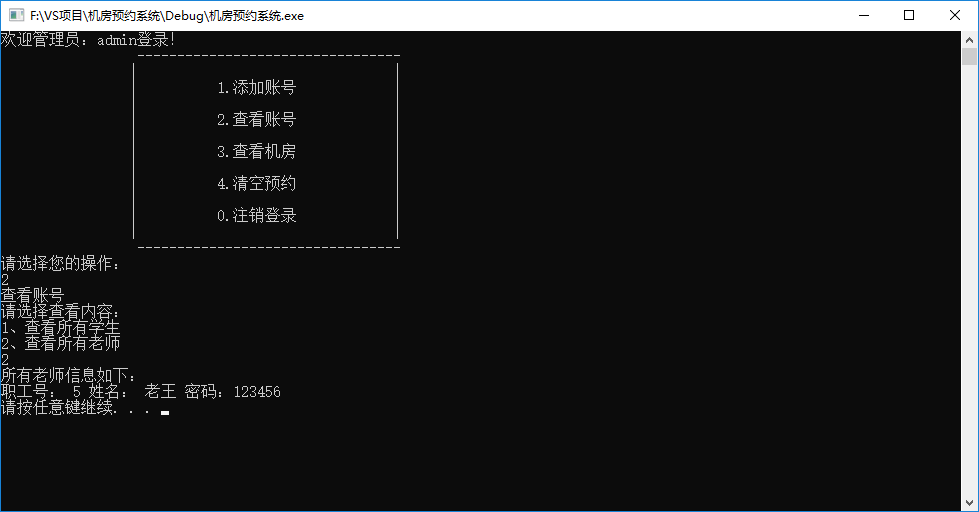
void printStudent(Student & s)  
{  
 cout << "学号： " << s.m\_Id << " 姓名： " << s.m\_Name << " 密码：" << s.m\_Pwd << endl;  
}  
void printTeacher(Teacher & t)  
{  
 cout << "职工号： " << t.m\_EmpId << " 姓名： " << t.m\_Name << " 密码：" << t.m\_Pwd << endl;  
}  
  
void Manager::showPerson()  
{  
 cout << "请选择查看内容：" << endl;  
 cout << "1、查看所有学生" << endl;  
 cout << "2、查看所有老师" << endl;  
  
 int select = 0;  
  
 cin >> select;  
   
 if (select == 1)  
 {  
 cout << "所有学生信息如下： " << endl;  
 for\_each(vStu.begin(), vStu.end(), printStudent);  
 }  
 else  
 {  
 cout << "所有老师信息如下： " << endl;  
 for\_each(vTea.begin(), vTea.end(), printTeacher);  
 }  
 system("pause");  
 system("cls");  
}

#### 7.3.2 测试

测试查看学生效果



测试查看教师效果



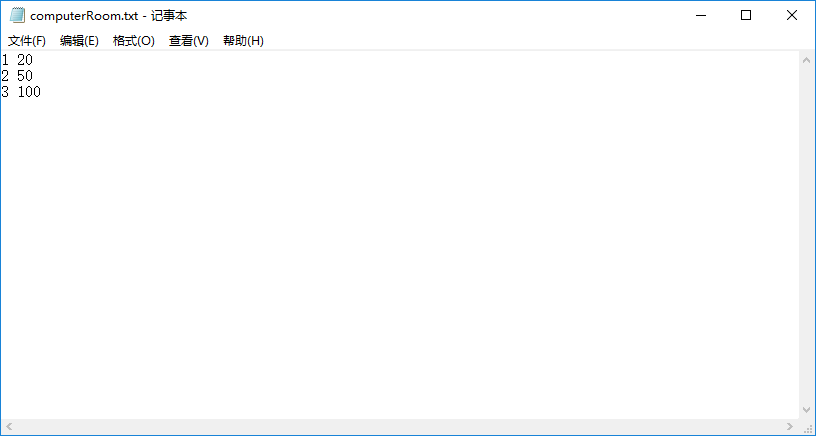
至此，显示账号功能实现完毕

### 7.4 查看机房

#### 7.4.1 添加机房信息

案例需求中，机房一共有三个，其中1号机房容量20台机器，2号50台，3号100台

我们可以将信息录入到computerRoom.txt中



#### 7.4.2 机房类创建

在头文件下，创建新的文件 computerRoom.h

并添加如下代码：

#pragma once  
#include<iostream>  
using namespace std;  
//机房类  
class ComputerRoom  
{  
public:  
  
 int m\_ComId; //机房id号  
  
 int m\_MaxNum; //机房最大容量  
};

#### 7.4.3 初始化机房信息

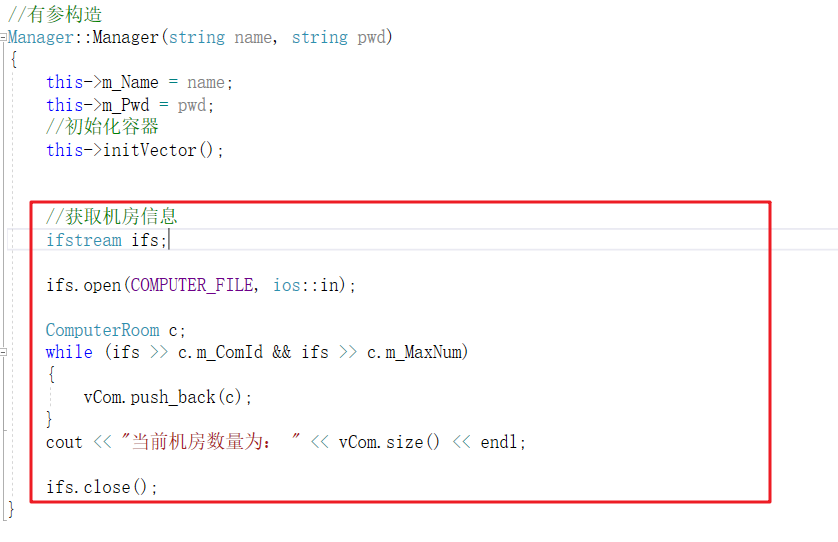
在Manager管理员类下，添加机房的容器,用于保存机房信息

//机房容器  
 vector<ComputerRoom> vCom;

在Manager有参构造函数中，追加如下代码，初始化机房信息

//获取机房信息  
 ifstream ifs;  
  
 ifs.open(COMPUTER\_FILE, ios::in);  
  
 ComputerRoom c;  
 while (ifs >> c.m\_ComId && ifs >> c.m\_MaxNum)  
 {  
 vCom.push\_back(c);  
 }  
 cout << "当前机房数量为： " << vCom.size() << endl;  
  
 ifs.close();

位置如图：



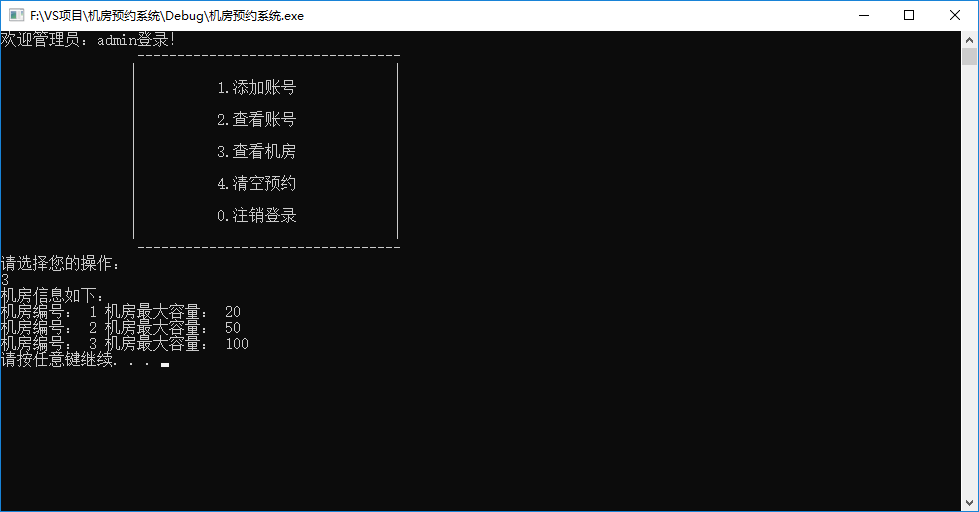
因为机房信息目前版本不会有所改动，如果后期有修改功能，最好封装到一个函数中，方便维护

#### 7.4.4 显示机房信息

在Manager类的showComputer成员函数中添加如下代码：

//查看机房信息  
void Manager::showComputer()  
{  
 cout << "机房信息如下： " << endl;  
 for (vector<ComputerRoom>::iterator it = vCom.begin(); it != vCom.end(); it++)  
 {  
 cout << "机房编号： " << it->m\_ComId << " 机房最大容量： " << it->m\_MaxNum << endl;  
 }  
 system("pause");  
 system("cls");  
}

测试显示机房信息功能：



### 7.5 清空预约

功能描述：

清空生成的order.txt预约文件

#### 7.5.1 清空功能实现

在Manager的cleanFile成员函数中添加如下代码：

//清空预约记录  
void Manager::cleanFile()  
{  
 ofstream ofs(ORDER\_FILE, ios::trunc);  
 ofs.close();  
  
 cout << "清空成功！" << endl;  
 system("pause");  
 system("cls");  
}

测试清空，可以随意写入一些信息在order.txt中，然后调用cleanFile清空文件接口，查看是否清空干净

## 8、 学生模块

### 8.1 学生登录和注销

#### 8.1.1 构造函数

* 在Student类的构造函数中，初始化学生信息，代码如下：

//有参构造(学号、姓名、密码)  
Student::Student(int id, string name, string pwd)  
{  
 //初始化属性  
 this->m\_Id = id;  
 this->m\_Name = name;  
 this->m\_Pwd = pwd;  
}

#### 8.1.2 管理员子菜单

* 在机房预约系统.cpp中，当用户登录的是学生，添加学生菜单接口
* 将不同的分支提供出来
  + 申请预约
  + 查看我的预约
  + 查看所有预约
  + 取消预约
  + 注销登录
* 实现注销功能

添加全局函数 void studentMenu(Identity \* &manager) 代码如下：

//学生菜单  
void studentMenu(Identity \* &student)  
{  
 while (true)  
 {  
 //学生菜单  
 student->operMenu();  
  
 Student\* stu = (Student\*)student;  
 int select = 0;  
  
 cin >> select;  
  
 if (select == 1) //申请预约  
 {  
 stu->applyOrder();  
 }  
 else if (select == 2) //查看自身预约  
 {  
 stu->showMyOrder();  
 }  
 else if (select == 3) //查看所有预约  
 {  
 stu->showAllOrder();  
 }  
 else if (select == 4) //取消预约  
 {  
 stu->cancelOrder();  
 }  
 else  
 {  
 delete student;  
 cout << "注销成功" << endl;  
 system("pause");  
 system("cls");  
 return;  
 }  
 }  
}

#### 8.1.3 菜单功能实现

* 在实现成员函数void Student::operMenu() 代码如下：

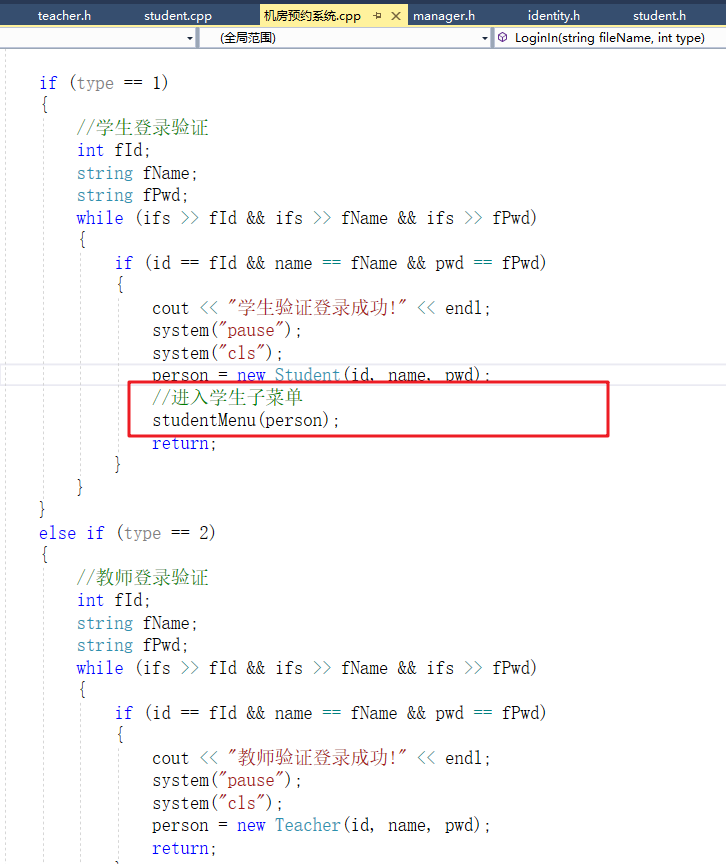
//菜单界面  
void Student::operMenu()  
{  
 cout << "欢迎学生代表：" << this->m\_Name << "登录！" << endl;  
 cout << "\t\t ----------------------------------\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 1.申请预约 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 2.查看我的预约 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 3.查看所有预约 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 4.取消预约 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 0.注销登录 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t ----------------------------------\n";  
 cout << "请选择您的操作： " << endl;  
}

#### 8.1.4 接口对接

* 学生成功登录后，调用学生的子菜单界面
* 在学生登录分支中，添加代码：

//进入学生子菜单  
 studentMenu(person);

添加效果如图：

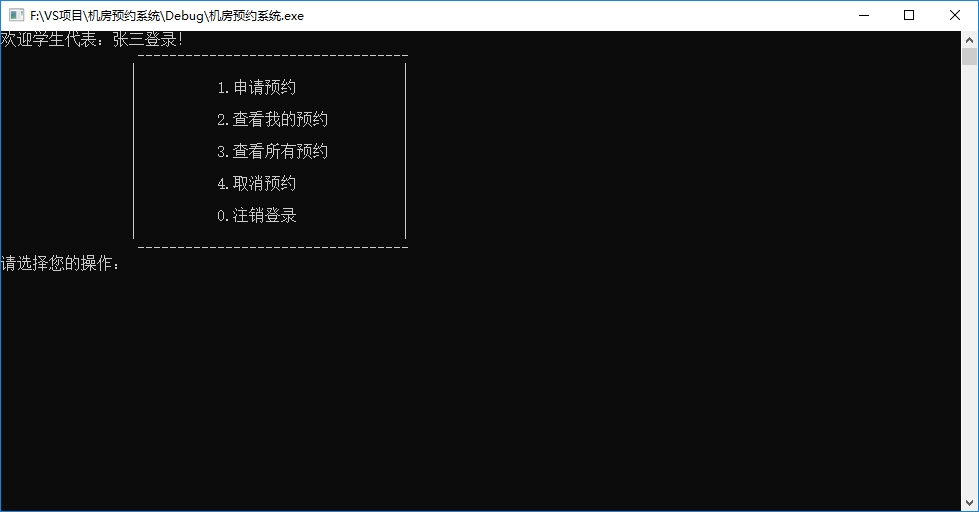


测试对接，效果如图：

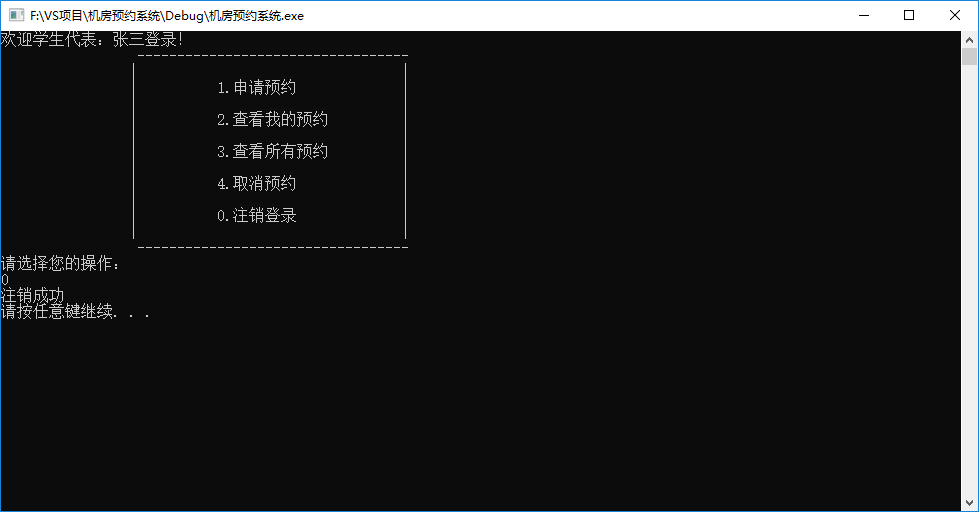
登录验证通过：



学生子菜单：



注销登录：



### 8.2 申请预约

#### 8.2.1 获取机房信息

* 在申请预约时，学生可以看到机房的信息，因此我们需要让学生获取到机房的信息

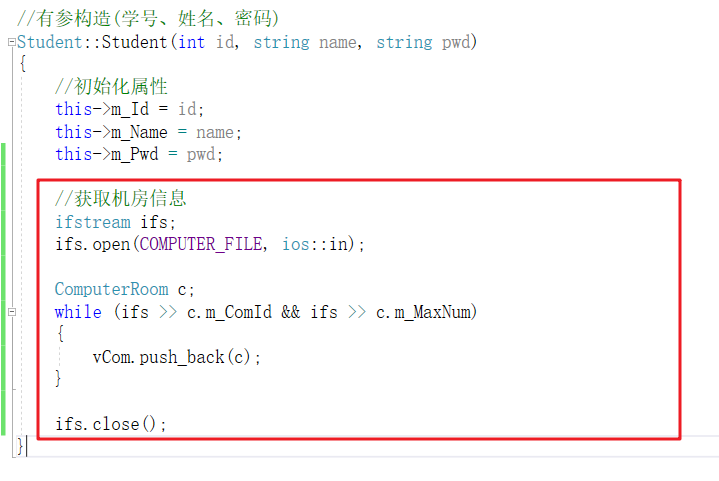
在student.h中添加新的成员函数如下：

//机房容器  
 vector<ComputerRoom> vCom;

在学生的有参构造函数中追加如下代码：

//获取机房信息  
 ifstream ifs;  
 ifs.open(COMPUTER\_FILE, ios::in);  
  
 ComputerRoom c;  
 while (ifs >> c.m\_ComId && ifs >> c.m\_MaxNum)  
 {  
 vCom.push\_back(c);  
 }  
  
 ifs.close();

追加位置如图：



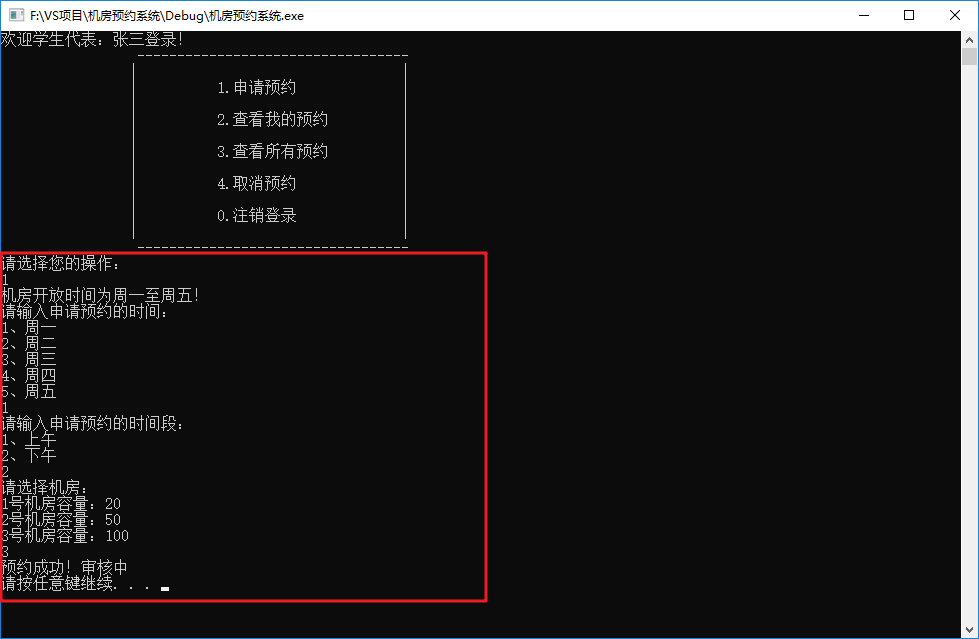
至此，vCom容器中保存了所有机房的信息

#### 8.2.2 预约功能实现

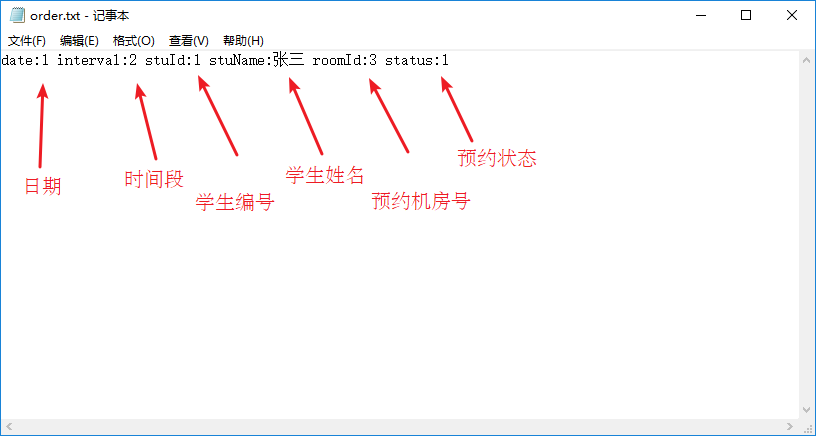
在student.cpp中实现成员函数 void Student::applyOrder()

//申请预约  
void Student::applyOrder()  
{  
 cout << "机房开放时间为周一至周五！" << endl;  
 cout << "请输入申请预约的时间：" << endl;  
 cout << "1、周一" << endl;  
 cout << "2、周二" << endl;  
 cout << "3、周三" << endl;  
 cout << "4、周四" << endl;  
 cout << "5、周五" << endl;  
 int date = 0;  
 int interval = 0;  
 int room = 0;  
  
 while (true)  
 {  
 cin >> date;  
 if (date >= 1 && date <= 5)  
 {  
 break;  
 }  
 cout << "输入有误，请重新输入" << endl;  
 }  
  
  
 cout << "请输入申请预约的时间段：" << endl;  
 cout << "1、上午" << endl;  
 cout << "2、下午" << endl;  
  
 while (true)  
 {  
 cin >> interval;  
 if (interval >= 1 && interval <= 2)  
 {  
 break;  
 }  
 cout << "输入有误，请重新输入" << endl;  
 }  
  
 cout << "请选择机房：" << endl;  
 cout << "1号机房容量：" << vCom[0].m\_MaxNum << endl;  
 cout << "2号机房容量：" << vCom[1].m\_MaxNum << endl;  
 cout << "3号机房容量：" << vCom[2].m\_MaxNum << endl;  
  
 while (true)  
 {  
 cin >> room;  
 if (room >= 1 && room <= 3)  
 {  
 break;  
 }  
 cout << "输入有误，请重新输入" << endl;  
 }  
  
 cout << "预约成功！审核中" << endl;  
  
 ofstream ofs(ORDER\_FILE, ios::app);  
 ofs << "date:" << date << " ";  
 ofs << "interval:" << interval << " ";  
 ofs << "stuId:" << this->m\_Id << " ";  
 ofs << "stuName:" << this->m\_Name << " ";  
 ofs << "roomId:" << room << " ";  
 ofs << "status:" << 1 << endl;  
  
 ofs.close();  
  
 system("pause");  
 system("cls");  
}

运行程序，测试代码:



在order.txt文件中生成如下内容：



### 8.3 显示预约

#### 8.3.1 创建预约类

功能描述：显示预约记录时，需要从文件中获取到所有记录，用来显示，创建预约的类来管理记录以及更新

在头文件以及源文件下分别创建**orderFile.h** 和 **orderFile.cpp**文件

orderFile.h中添加如下代码：

#pragma once  
#include<iostream>  
using namespace std;  
#include <map>  
#include "globalFile.h"  
  
class OrderFile  
{  
public:  
  
 //构造函数  
 OrderFile();  
  
 //更新预约记录  
 void updateOrder();  
  
 //记录的容器 key --- 记录的条数 value --- 具体记录的键值对信息  
 map<int, map<string, string>> m\_orderData;  
  
 //预约记录条数  
 int m\_Size;  
};

**构造函数**中获取所有信息，并存放在容器中，添加如下代码：

OrderFile::OrderFile()  
{  
 ifstream ifs;  
 ifs.open(ORDER\_FILE, ios::in);  
  
 string date; //日期  
 string interval; //时间段  
 string stuId; //学生编号  
 string stuName; //学生姓名  
 string roomId; //机房编号  
 string status; //预约状态  
  
  
 this->m\_Size = 0; //预约记录个数  
  
 while (ifs >> date && ifs >> interval && ifs >> stuId && ifs >> stuName && ifs >> roomId && ifs >> status)  
 {  
 //测试代码  
 /\*  
 cout << date << endl;  
 cout << interval << endl;  
 cout << stuId << endl;  
 cout << stuName << endl;  
 cout << roomId << endl;  
 cout << status << endl;  
 \*/  
  
 string key;  
 string value;  
 map<string, string> m;  
  
 int pos = date.find(":");  
 if (pos != -1)  
 {  
 key = date.substr(0, pos);  
 value = date.substr(pos + 1, date.size() - pos -1);  
 m.insert(make\_pair(key, value));  
 }  
  
 pos = interval.find(":");  
 if (pos != -1)  
 {  
 key = interval.substr(0, pos);  
 value = interval.substr(pos + 1, interval.size() - pos -1 );  
 m.insert(make\_pair(key, value));  
 }  
  
 pos = stuId.find(":");  
 if (pos != -1)  
 {  
 key = stuId.substr(0, pos);  
 value = stuId.substr(pos + 1, stuId.size() - pos -1 );  
 m.insert(make\_pair(key, value));  
 }  
  
 pos = stuName.find(":");  
 if (pos != -1)  
 {  
 key = stuName.substr(0, pos);  
 value = stuName.substr(pos + 1, stuName.size() - pos -1);  
 m.insert(make\_pair(key, value));  
 }  
  
 pos = roomId.find(":");  
 if (pos != -1)  
 {  
 key = roomId.substr(0, pos);  
 value = roomId.substr(pos + 1, roomId.size() - pos -1 );  
 m.insert(make\_pair(key, value));  
 }  
  
 pos = status.find(":");  
 if (pos != -1)  
 {  
 key = status.substr(0, pos);  
 value = status.substr(pos + 1, status.size() - pos -1);  
 m.insert(make\_pair(key, value));  
 }  
  
  
 this->m\_orderData.insert(make\_pair(this->m\_Size, m));  
 this->m\_Size++;  
 }  
  
 //测试代码  
 //for (map<int, map<string, string>>::iterator it = m\_orderData.begin(); it != m\_orderData.end();it++)  
 //{  
 // cout << "key = " << it->first << " value = " << endl;  
 // for (map<string, string>::iterator mit = it->second.begin(); mit != it->second.end(); mit++)  
 // {  
 // cout << mit->first << " " << mit->second << " ";  
 // }  
 // cout << endl;  
 //}  
   
 ifs.close();  
}

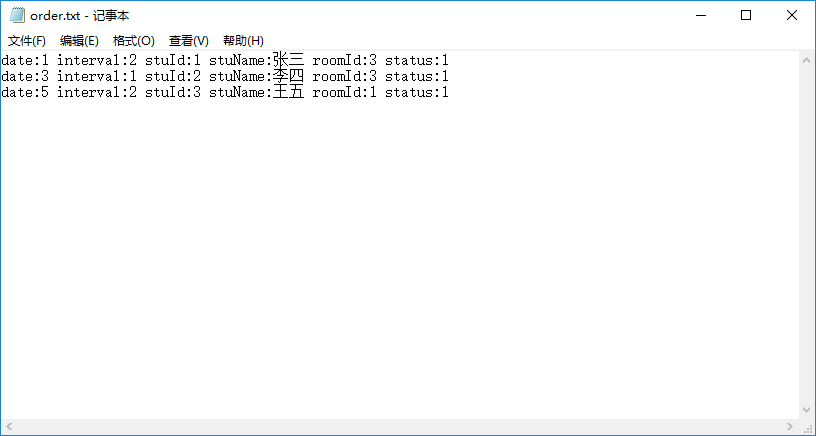
更新预约记录的成员函数updateOrder代码如下：

void OrderFile::updateOrder()  
{  
 if (this->m\_Size == 0)  
 {  
 return;  
 }  
  
 ofstream ofs(ORDER\_FILE, ios::out | ios::trunc);  
 for (int i = 0; i < m\_Size;i++)  
 {  
 ofs << "date:" << this->m\_orderData[i]["date"] << " ";  
 ofs << "interval:" << this->m\_orderData[i]["interval"] << " ";  
 ofs << "stuId:" << this->m\_orderData[i]["stuId"] << " ";  
 ofs << "stuName:" << this->m\_orderData[i]["stuName"] << " ";  
 ofs << "roomId:" << this->m\_orderData[i]["roomId"] << " ";  
 ofs << "status:" << this->m\_orderData[i]["status"] << endl;  
 }  
 ofs.close();  
}

#### 8.3.2 显示自身预约

首先我们先添加几条预约记录，可以用程序添加或者直接修改order.txt文件

order.txt文件内容如下： 比如我们有三名同学分别产生了3条预约记录



在Student类的void Student::showMyOrder()成员函数中，添加如下代码

//查看我的预约  
void Student::showMyOrder()  
{  
 OrderFile of;  
 if (of.m\_Size == 0)  
 {  
 cout << "无预约记录" << endl;  
 system("pause");  
 system("cls");  
 return;  
 }  
 for (int i = 0; i < of.m\_Size; i++)  
 {  
 if (atoi(of.m\_orderData[i]["stuId"].c\_str()) == this->m\_Id)  
 {  
 cout << "预约日期： 周" << of.m\_orderData[i]["date"];  
 cout << " 时段：" << (of.m\_orderData[i]["interval"] == "1" ? "上午" : "下午");  
 cout << " 机房：" << of.m\_orderData[i]["roomId"];  
 string status = " 状态： "; // 0 取消的预约 1 审核中 2 已预约 -1 预约失败  
 if (of.m\_orderData[i]["status"] == "1")  
 {  
 status += "审核中";  
 }  
 else if (of.m\_orderData[i]["status"] == "2")  
 {  
 status += "预约成功";  
 }  
 else if (of.m\_orderData[i]["status"] == "-1")  
 {  
 status += "审核未通过，预约失败";  
 }  
 else  
 {  
 status += "预约已取消";  
 }  
 cout << status << endl;  
  
 }  
 }  
  
 system("pause");  
 system("cls");  
}

测试效果如图：



#### 8.3.3 显示所有预约

在Student类的void Student::showAllOrder()成员函数中，添加如下代码

//查看所有预约  
void Student::showAllOrder()  
{  
 OrderFile of;  
 if (of.m\_Size == 0)  
 {  
 cout << "无预约记录" << endl;  
 system("pause");  
 system("cls");  
 return;  
 }  
  
 for (int i = 0; i < of.m\_Size; i++)  
 {  
 cout << i + 1 << "、 ";  
  
 cout << "预约日期： 周" << of.m\_orderData[i]["date"];  
 cout << " 时段：" << (of.m\_orderData[i]["interval"] == "1" ? "上午" : "下午");  
 cout << " 学号：" << of.m\_orderData[i]["stuId"];  
 cout << " 姓名：" << of.m\_orderData[i]["stuName"];  
 cout << " 机房：" << of.m\_orderData[i]["roomId"];  
 string status = " 状态： "; // 0 取消的预约 1 审核中 2 已预约 -1 预约失败  
 if (of.m\_orderData[i]["status"] == "1")  
 {  
 status += "审核中";  
 }  
 else if (of.m\_orderData[i]["status"] == "2")  
 {  
 status += "预约成功";  
 }  
 else if (of.m\_orderData[i]["status"] == "-1")  
 {  
 status += "审核未通过，预约失败";  
 }  
 else  
 {  
 status += "预约已取消";  
 }  
 cout << status << endl;  
 }  
  
 system("pause");  
 system("cls");  
}

测试效果如图：

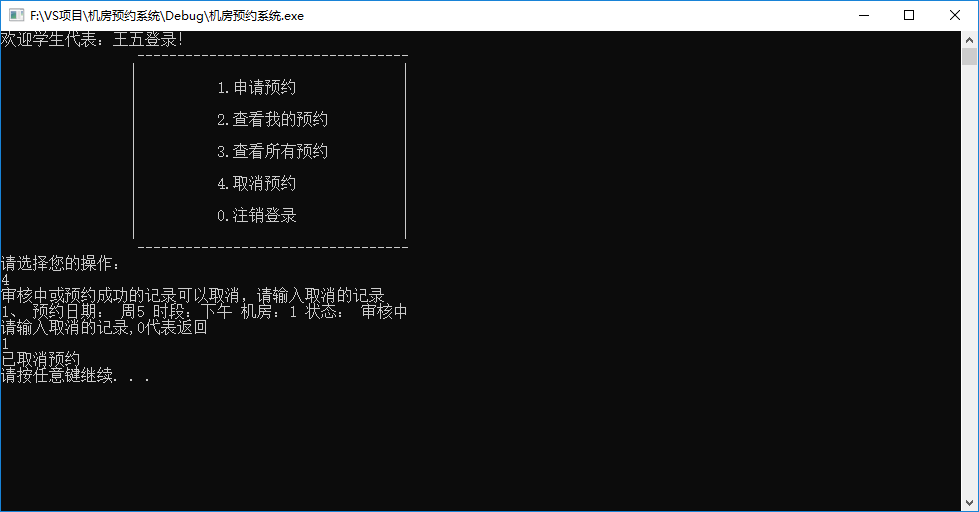


### 8.4 取消预约

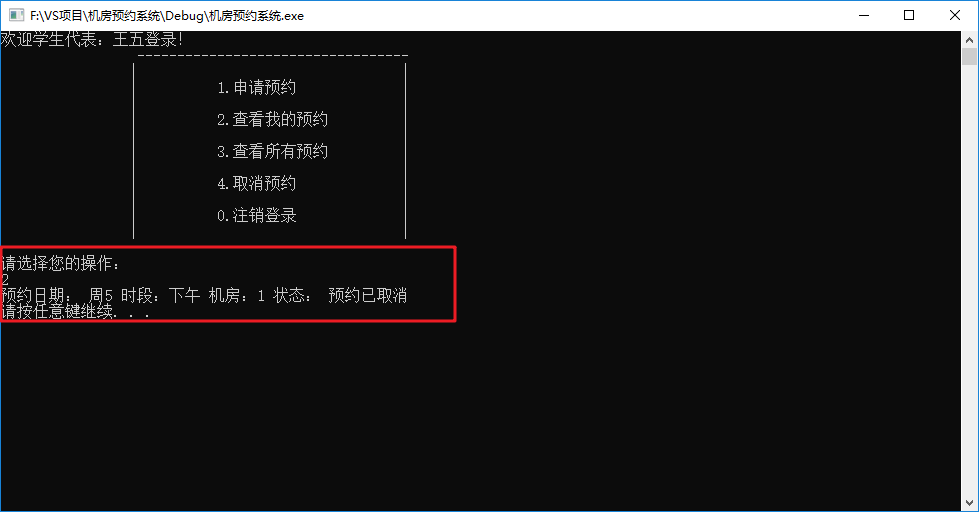
在Student类的void Student::cancelOrder()成员函数中，添加如下代码

//取消预约  
void Student::cancelOrder()  
{  
 OrderFile of;  
 if (of.m\_Size == 0)  
 {  
 cout << "无预约记录" << endl;  
 system("pause");  
 system("cls");  
 return;  
 }  
 cout << "审核中或预约成功的记录可以取消，请输入取消的记录" << endl;  
  
 vector<int>v;  
 int index = 1;  
 for (int i = 0; i < of.m\_Size; i++)  
 {  
 if (atoi(of.m\_orderData[i]["stuId"].c\_str()) == this->m\_Id)  
 {  
 if (of.m\_orderData[i]["status"] == "1" || of.m\_orderData[i]["status"] == "2")  
 {  
 v.push\_back(i);  
 cout << index ++ << "、 ";  
 cout << "预约日期： 周" << of.m\_orderData[i]["date"];  
 cout << " 时段：" << (of.m\_orderData[i]["interval"] == "1" ? "上午" : "下午");  
 cout << " 机房：" << of.m\_orderData[i]["roomId"];  
 string status = " 状态： "; // 0 取消的预约 1 审核中 2 已预约 -1 预约失败  
 if (of.m\_orderData[i]["status"] == "1")  
 {  
 status += "审核中";  
 }  
 else if (of.m\_orderData[i]["status"] == "2")  
 {  
 status += "预约成功";  
 }  
 cout << status << endl;  
  
 }  
 }  
 }  
  
 cout << "请输入取消的记录,0代表返回" << endl;  
 int select = 0;  
 while (true)  
 {  
 cin >> select;  
 if (select >= 0 && select <= v.size())  
 {  
 if (select == 0)  
 {  
 break;  
 }  
 else  
 {  
 // cout << "记录所在位置： " << v[select - 1] << endl;  
 of.m\_orderData[v[select - 1]]["status"] = "0";  
 of.updateOrder();  
 cout << "已取消预约" << endl;  
 break;  
 }  
  
 }  
 cout << "输入有误，请重新输入" << endl;  
 }  
  
 system("pause");  
 system("cls");  
}

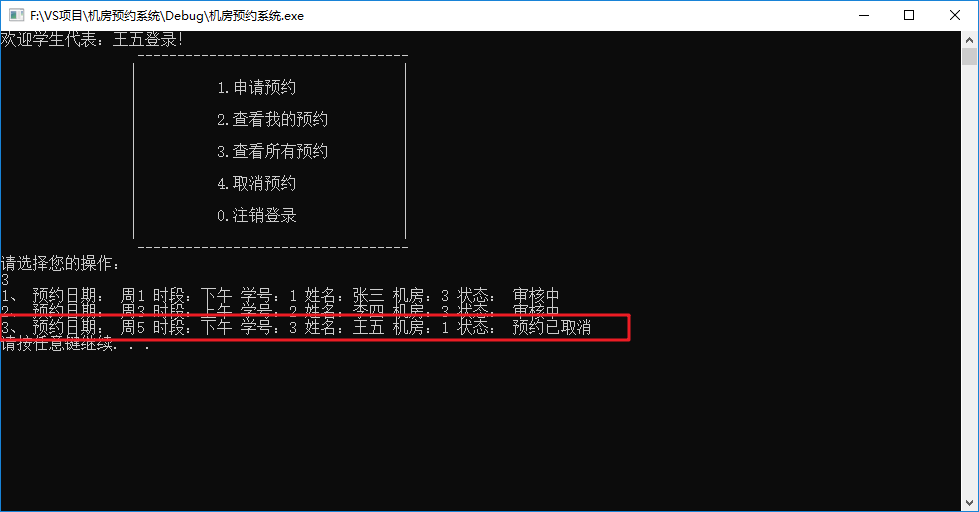
测试取消预约：



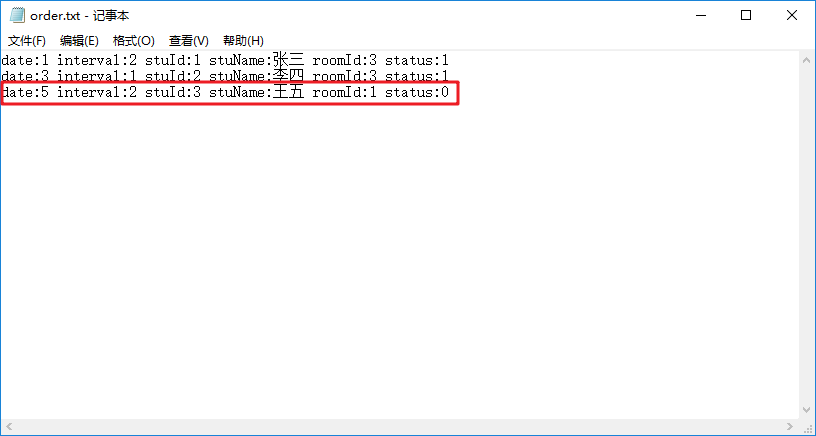
再次查看个人预约记录：



查看所有预约



查看order.txt预约文件



至此，学生模块功能全部实现

## 9、 教师模块

### 9.1 教师登录和注销

#### 9.1.1 构造函数

* 在Teacher类的构造函数中，初始化教师信息，代码如下：

//有参构造 (职工编号，姓名，密码)  
Teacher::Teacher(int empId, string name, string pwd)  
{  
 //初始化属性  
 this->m\_EmpId = empId;  
 this->m\_Name = name;  
 this->m\_Pwd = pwd;  
}

#### 9.1.2 教师子菜单

* 在机房预约系统.cpp中，当用户登录的是教师，添加教师菜单接口
* 将不同的分支提供出来
  + 查看所有预约
  + 审核预约
  + 注销登录
* 实现注销功能

添加全局函数 void TeacherMenu(Person \* &manager) 代码如下：

//教师菜单  
void TeacherMenu(Identity \* &teacher)  
{  
 while (true)  
 {  
 //教师菜单  
 teacher->operMenu();  
  
 Teacher\* tea = (Teacher\*)teacher;  
 int select = 0;  
  
 cin >> select;  
  
 if (select == 1)  
 {  
 //查看所有预约  
 tea->showAllOrder();  
 }  
 else if (select == 2)  
 {  
 //审核预约  
 tea->validOrder();  
 }  
 else  
 {  
 delete teacher;  
 cout << "注销成功" << endl;  
 system("pause");  
 system("cls");  
 return;  
 }  
  
 }  
}

#### 9.1.3 菜单功能实现

* 在实现成员函数void Teacher::operMenu() 代码如下：

//教师菜单界面  
void Teacher::operMenu()  
{  
 cout << "欢迎教师：" << this->m\_Name << "登录！" << endl;  
 cout << "\t\t ----------------------------------\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 1.查看所有预约 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 2.审核预约 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t| 0.注销登录 |\n";  
 cout << "\t\t| |\n";  
 cout << "\t\t ----------------------------------\n";  
 cout << "请选择您的操作： " << endl;  
}

#### 9.1.4 接口对接

* 教师成功登录后，调用教师的子菜单界面
* 在教师登录分支中，添加代码：

//进入教师子菜单  
 TeacherMenu(person);

添加效果如图：



测试对接，效果如图：

登录验证通过：



教师子菜单：



注销登录：



### 9.2 查看所有预约

#### 9.2.1 所有预约功能实现

该功能与学生身份的查看所有预约功能相似，用于显示所有预约记录

在Teacher.cpp中实现成员函数 void Teacher::showAllOrder()

void Teacher::showAllOrder()  
{  
 OrderFile of;  
 if (of.m\_Size == 0)  
 {  
 cout << "无预约记录" << endl;  
 system("pause");  
 system("cls");  
 return;  
 }  
 for (int i = 0; i < of.m\_Size; i++)  
 {  
 cout << i + 1 << "、 ";  
  
 cout << "预约日期： 周" << of.m\_orderData[i]["date"];  
 cout << " 时段：" << (of.m\_orderData[i]["interval"] == "1" ? "上午" : "下午");  
 cout << " 学号：" << of.m\_orderData[i]["stuId"];  
 cout << " 姓名：" << of.m\_orderData[i]["stuName"];  
 cout << " 机房：" << of.m\_orderData[i]["roomId"];  
 string status = " 状态： "; // 0 取消的预约 1 审核中 2 已预约 -1 预约失败  
 if (of.m\_orderData[i]["status"] == "1")  
 {  
 status += "审核中";  
 }  
 else if (of.m\_orderData[i]["status"] == "2")  
 {  
 status += "预约成功";  
 }  
 else if (of.m\_orderData[i]["status"] == "-1")  
 {  
 status += "审核未通过，预约失败";  
 }  
 else  
 {  
 status += "预约已取消";  
 }  
 cout << status << endl;  
 }  
  
 system("pause");  
 system("cls");  
}

#### 9.2.2 测试功能

运行测试教师身份的查看所有预约功能

测试效果如图：



### 9.3 审核预约

#### 9.3.1 审核功能实现

功能描述：教师审核学生的预约，依据实际情况审核预约

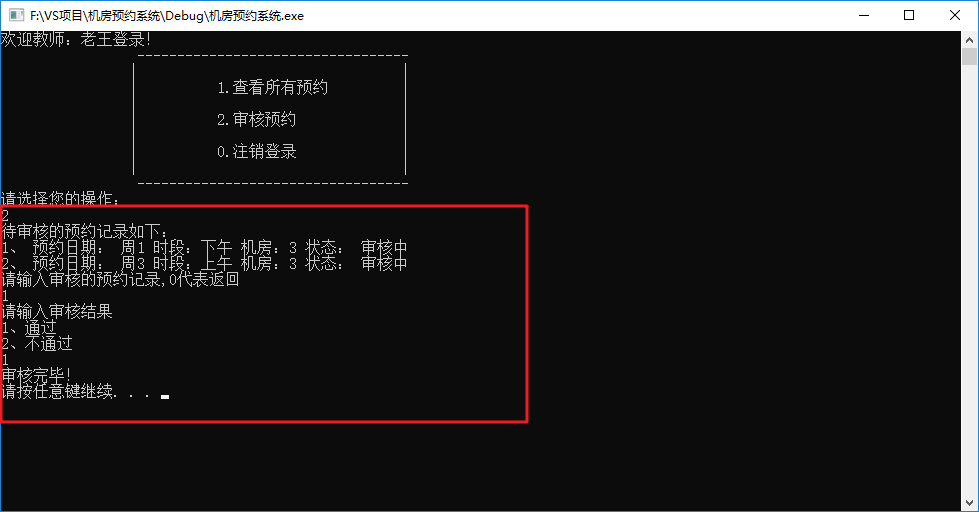
在Teacher.cpp中实现成员函数 void Teacher::validOrder()

代码如下：

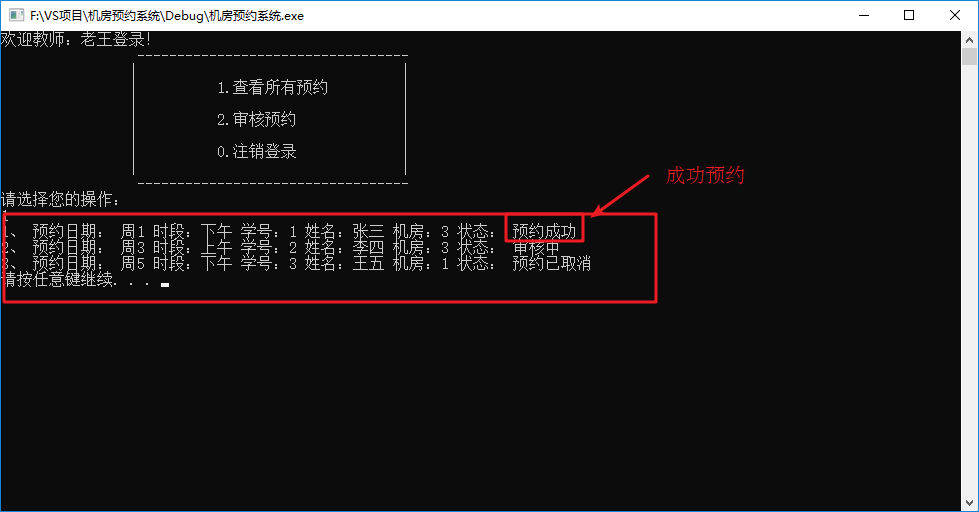
//审核预约  
void Teacher::validOrder()  
{  
 OrderFile of;  
 if (of.m\_Size == 0)  
 {  
 cout << "无预约记录" << endl;  
 system("pause");  
 system("cls");  
 return;  
 }  
 cout << "待审核的预约记录如下：" << endl;  
  
 vector<int>v;  
 int index = 0;  
 for (int i = 0; i < of.m\_Size; i++)  
 {  
 if (of.m\_orderData[i]["status"] == "1")  
 {  
 v.push\_back(i);  
 cout << ++index << "、 ";  
 cout << "预约日期： 周" << of.m\_orderData[i]["date"];  
 cout << " 时段：" << (of.m\_orderData[i]["interval"] == "1" ? "上午" : "下午");  
 cout << " 机房：" << of.m\_orderData[i]["roomId"];  
 string status = " 状态： "; // 0取消的预约 1 审核中 2 已预约 -1 预约失败  
 if (of.m\_orderData[i]["status"] == "1")  
 {  
 status += "审核中";  
 }  
 cout << status << endl;  
 }  
 }  
 cout << "请输入审核的预约记录,0代表返回" << endl;  
 int select = 0;  
 int ret = 0;  
 while (true)  
 {  
 cin >> select;  
 if (select >= 0 && select <= v.size())  
 {  
 if (select == 0)  
 {  
 break;  
 }  
 else  
 {  
 cout << "请输入审核结果" << endl;  
 cout << "1、通过" << endl;  
 cout << "2、不通过" << endl;  
 cin >> ret;  
  
 if (ret == 1)  
 {  
 of.m\_orderData[v[select - 1]]["status"] = "2";  
 }  
 else  
 {  
 of.m\_orderData[v[select - 1]]["status"] = "-1";  
 }  
 of.updateOrder();  
 cout << "审核完毕！" << endl;  
 break;  
 }  
 }  
 cout << "输入有误，请重新输入" << endl;  
 }  
  
 system("pause");  
 system("cls");  
}

#### 9.3.2 测试审核预约

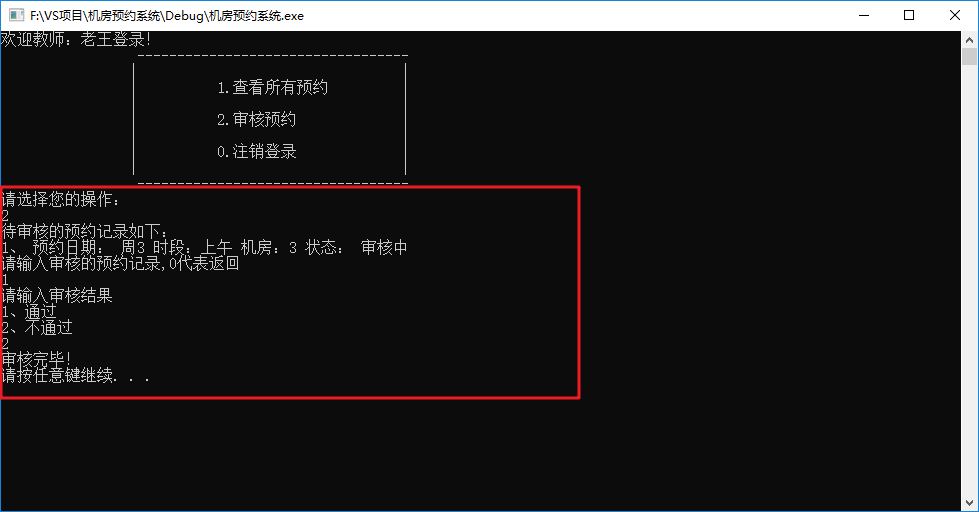
测试 - 审核通过



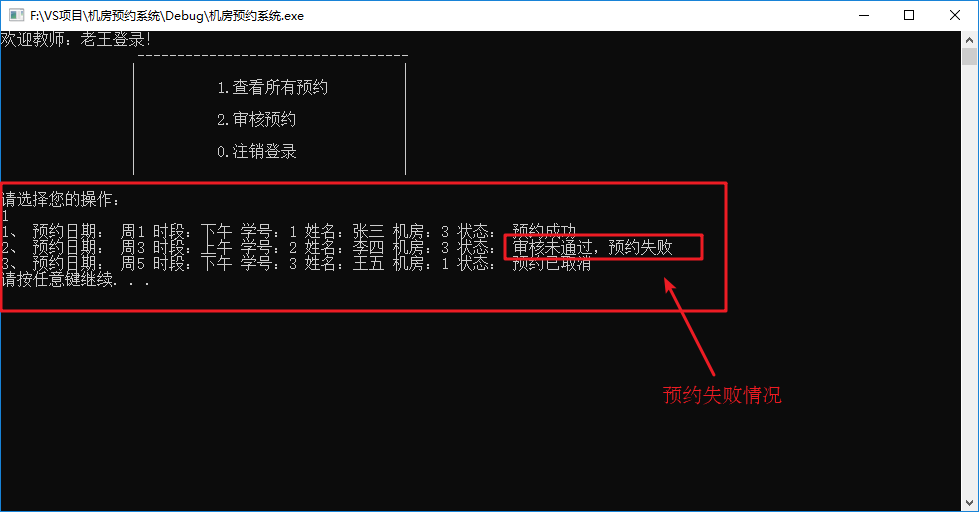
审核通过情况



测试-审核未通过



审核未通过情况：



学生身份下查看记录：



审核预约成功！

至此本案例制作完毕！ ^\_^