实验报告（三）

#### 一、实验目的

(1) 掌握类的声明和使用

(2) 掌握类的声明和对象的声明

(3) 复习具有不同访问属性的成员的访问方式

(4) 观察构造函数和析构函数的执行过程

(5) 学习类的组合使用方法

(6) 观察程序运行中变量的作用域、生存期和可见性。

(7) 学习类的静态成员的使用。

(8) 学习多文件结构在C++程序中的使用。

(9) 学会使用Visual Studio 2008 开发环境中的debuge调试功能：单步执行、设置断点、观察变量值。

#### 二、实验内容

1、运行以下程序，观察程序输出。

#include<iostream>

using namespace std;

void fn1();

int x=1,y=2;

int main()

{

cout<<"Begin..."<<endl;

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

cout<<"y="<<y<<endl;

cout<<"Evaluate x and y in main()..."<<endl;

int x=10,y=20;

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

cout<<"y="<<y<<endl;

cout<<"Step int fn1()..."<<endl;

fn1();

cout<<"Back in main"<<endl;

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

cout<<"y="<<y<<endl;

return 0;

}

void fn1()

{

int y=200;

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

cout<<"y="<<y<<endl;

}

2、新建一个空的项目lab5\_2,添加头文件client.h，在其中声明类CLIENT,注意使用编译预处理命令；再添加源程序文件client.cpp,在其中实现CLIENT类，注意静态成员变量的使用方法；再添加文件lab5\_2.cpp,在其中定义main（）函数，测试CLIENT类，观察相应的成员变量取值的变化情况

#### 三、实验环境

硬件环境：AMD A12；操作系统：Windows 10；C++环境：Visual Studio 2017

#### 四、程序设计原理和方法

##### 1、第一个小实验：观察程序输出

由于此实验的实验代码已经给出，因此在此部分，我来分析这段代码的设计原理和方法。

首先，代码声明了二函数fn，并在后面给出了此函数的定义，并在fn的函数体中又定义了y，它只在fn中起作用，具有局部作用域。

然后，声明并定义了全局变量x，y，它们具有静态生存期。

接着，在主函数中，又再次定义了x，y，此时他们作为局部变量，具有动态生存期，其作用域只为主函数。

然后输出具有不同作用域和生存期的x和y，观察他们的不同。

##### 2、第二个小实验的设计

我设计这个程序是为了在位置有限的情况下，帮助客户预约，并把这些程序按照功能的不同分写在一个c++项目的三个文件中。

首先，根据题意建立头文件client.h，定义CLIENT类。在这里，我设置了四个数据成员，其中三个是动态成员变量，分别表示客户的基本信息：ID，年龄和预约时间；另外一个是静态数据成员number，用于表示可预订的座位数量（因为可预订的座位数量与客户本身无关，所以将其声明为全局变量）。

我还设置了四个函数成员。其中三个具有动态生存期，为了得到客户的基本信息（ID，年龄，预约时间），另一个具有静态生存期，为了表示还剩下多少座位。

然后，建立源文件client.cpp实现头文件中的类CLIENT，主要是对CLIENT中的函数成员进行定义与解释。

最后，编写主函数代码，告知客户还剩下多少座位，并得到具体的客户信息，帮助客户预约。

#### 五、程序代码

##### 1、第一个小实验

不需要写程序代码

##### 2、第二个小实验

###### （1）头文件：client.h

#pragma once

class CLIENT {

public:

CLIENT(int ID, int age, int time) {

this->ID = ID;

this->age = age;

this->time = time;

}

void getTime();

void getAge();

void getID();

static void show();

private:

int ID, age, time;

static int number;

};

###### （2）源文件：client.cpp

#include"client.h"

#include<iostream>

using namespace std;

int CLIENT::number = 20;

void CLIENT::getTime() { cout << "Time:" << time << endl; }

void CLIENT::getAge() { cout << "Age:" << age << endl; }

void CLIENT::getID() { cout << "ID:" << ID << endl; }

void CLIENT::show() {

cout << "There are only " << number << " available." << endl;

}

###### （3）主函数：label5\_2.cpp

#include"client.h"

#include<iostream>

using namespace std;

int main() {

int ID, age, time;

cout << "Please enter some information and we will help you make an appointment!" << endl;

cout << "Please enter your ID number:";

cin >> ID;

cout << "Please enter your age:";

cin >> age;

cout << "Please enter the time you want to be here:";

cin >> time;

CLIENT client(ID, age, time);

client.show();

cout << "Your information is here:" << endl;

client.getID();

client.getAge();

client.getTime();

system("pause");

return 0;

}

#### 六、运行结果

##### 1、第一个小实验

Begin...

x=1

y=2

Evaluate x and y in main()...

x=10

y=20

Step int fn1()...

x=1

y=200

Back in main

x=10

y=20

##### 2、第一个小实验

Please enter some information and we will help you make an appointment!

Please enter your ID number:256

Please enter your age:19

Please enter the time you want to be here:15

There are only 20 available.

Your information is here:

ID:256

Age:19

Time:15

#### 七、解释运行结果

##### 1、第一个小实验

输出的第一组x与y的值是全局变量x与y的值，分别为1和2.

输出的第二组x和y的值是在主函数中新定义的x与y的值，分别为10和20.因为主函数中重新定义了x和y作为局部变量，覆盖了一开始的全局变量x和y，因而在主函数中，x与y的值被改变了。

输出的第三组x和y是调用fn的时候输出的x和y的值，分别为1和200.x=1是因为fn函数中没有重新定义fn，因此还是使用全局变量x的值1，y=200是因为fn函数中重新定义y=200作为局部变量，覆盖了全局变量的值y=2。

输出的第四组x与y的值仍是主函数中x与y的值，分别为10和20。

##### 2、第二个小实验

根据提示输入的三个数分别作为客户的ID，年龄和预约时间。

首先向调用client.show();得到“There are only 20 available.”剩下的座位数。

然后调用getID，getAge，getTime来得到预约的基本信息，于是输出了“ID:256”“Age:19”“Time:15”。

#### 八、实验小结

##### 1、第一个小实验

在观察输出结果时，我采用了设置断点的逐语句调试方式，清楚地理解了局部变量和全局变量之间的关系，它们的生存期以及覆盖是什么样子的：全局变量的生存期是静态的，而局部变量的生存期是动态的，会随着所在作用域的消失而消失。此外，同名的局部变量也会覆盖全局变量。

通过这个实验，我加深了对数据共享和保护这一章节的理解。

##### 2、第二个小实验

由于此实验运用到了编译预处理，将类的定义，类的实现和主函数分别写在了不同的文件中，所以让我更加明确了每一个文件的作用，也让我对代码结构的理解变得更加清晰了。

同时，此项目中包含了静态成员，也让我明白了静态成员在这些文件中都是共享的。