电子科技大学信息与软件工程学院

**实验报告**

点名册序号 304-3 学 号 2015220103022

姓 名 张健顺

（实验） 课程名称 面向对象程序设计(C++)

理论教师 陈安龙

实验教师 陈安龙

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：张健顺 学号：2015220103022 指导教师：陈安龙**

**实验地点：信软楼西304 实验时间：16.11.08**

1. **实验名称：继承与派生程序设计**
2. **实验学时：4**
3. **实验目的：**

通过实验练习，理解继承与派生的概念及意义；掌握派生类的定义方法和派生类构造函数的定义方法；掌握不同继承方式对基类成员在派生类中访问权限的影响；掌握派生类对象的构造与析构过程。

1. **实验内容：**

利用继承与派生机制实现学校的学生与教师基本信息管理。假设各类人员的基本信息如下：

Teacher（教师）类：姓名、年龄、性别、工号、职称；

Undergraduate（本科生）类：姓名、年龄、性别、学号、系别

Postgraduate（研究生）类：姓名、年龄、性别、学号、系别、导师

根据上面的要求完成如下任务：

设计基类Person，包含各类人员共有的基本信息，以及静态数据成员counter，用于统计学校人员总数；设计设置人员基本信息、显示人员基本信息及获取人员总数的成员函数。Person类声明如下：

class Person

{

protected:

string name;

int age;

char gender;

static int counter； // 统计总人数

public:

Person();

Person(string, int, char); //通过参数初始化成员

~Person();

void set(string, int, char); //设置人员基本信息

void show(); //显示人员基本信息

int get\_counter(); //获取人员总数

}；

（2）编程实现Person类各成员函数；分析其余各类人员的特征，设计合理的继承结构，并在Person类的基础上派生出其余各类人员，每个类需要有构造函数和析构函数、设置人员信息、显示人员基本信息及获取人员总数的成员函数。

（3）分别采用public、protected、private 三种继承方式，观察不同继承方式对基类成员在派生类中访问属性的影响。

（4）在主函数中测试各类的功能。

要求：

（1）理解继承与派生的概念；

（2）掌握派生类的定义方法和派生类构造函数的定义方法;

（3）掌握不同继承方式对基类成员在派生类中访问权限的影响；

（4）掌握派生类对象的构造与析构过程；

（5）掌握静态数据成员的继承方法。

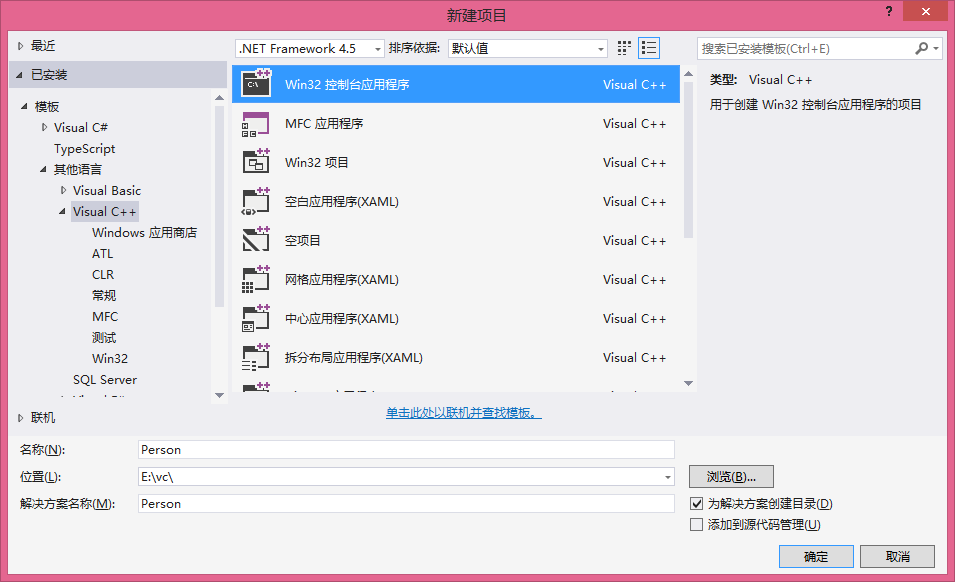
（6）掌握基类同名成员的访问方法。

1. **实验器材（设备、元器件）：**

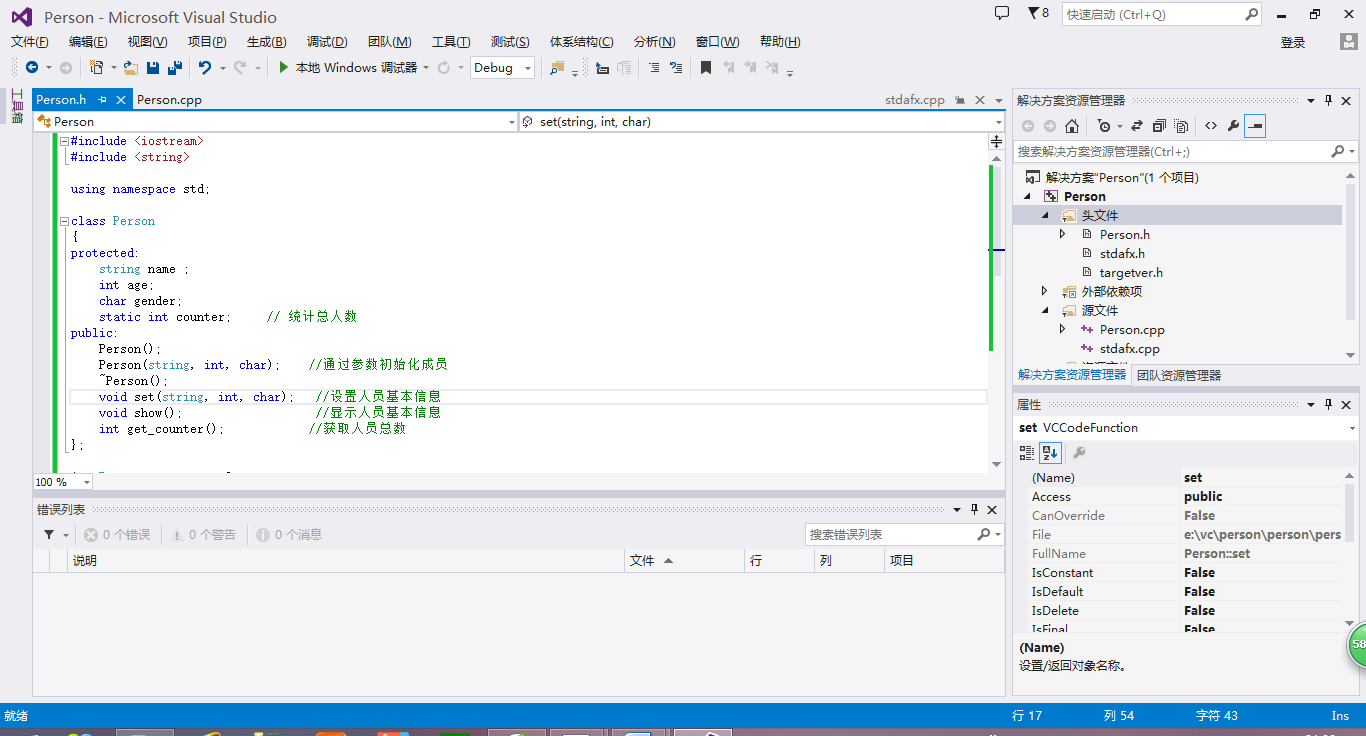
PC机一台，Windows 8.1，Visual Studio 2013

**六、 实验步骤：**

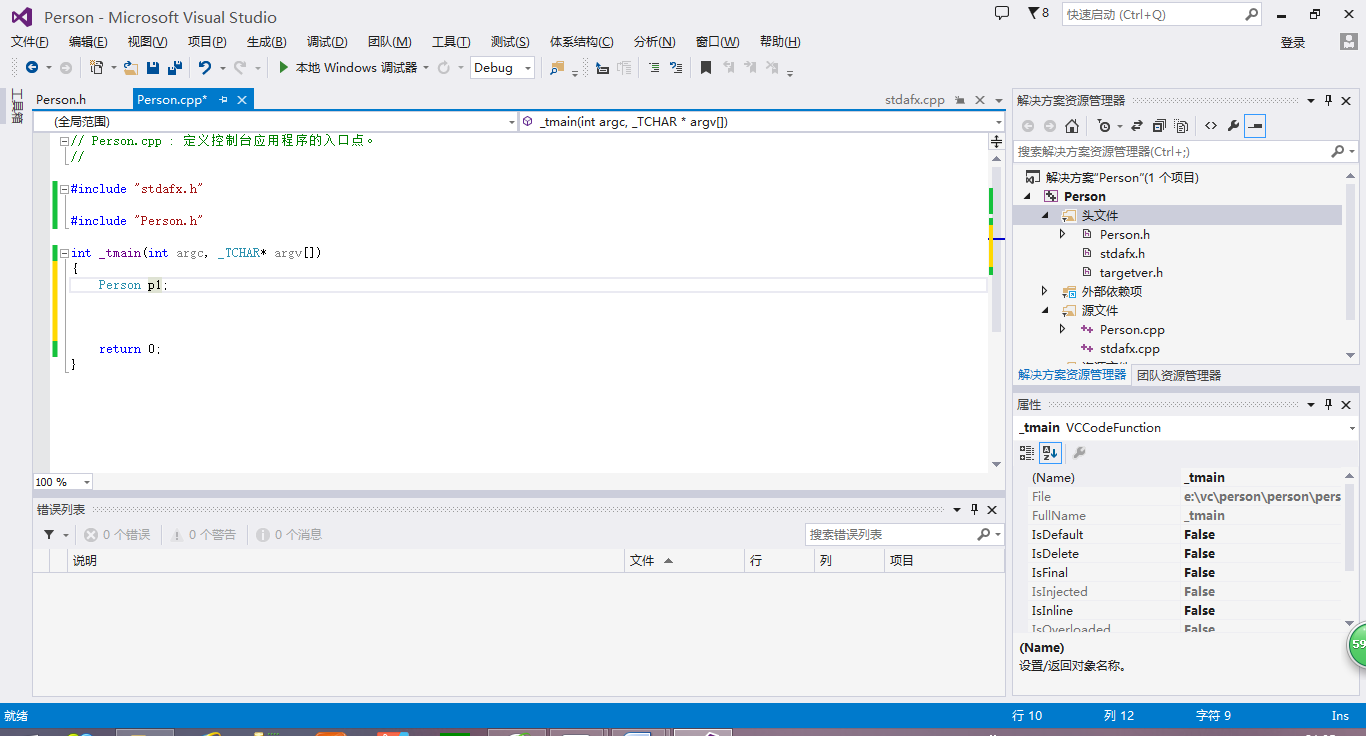
(1) 建立工程，项目名称设置为“Person”，类型选择“Win32控制台应用”。



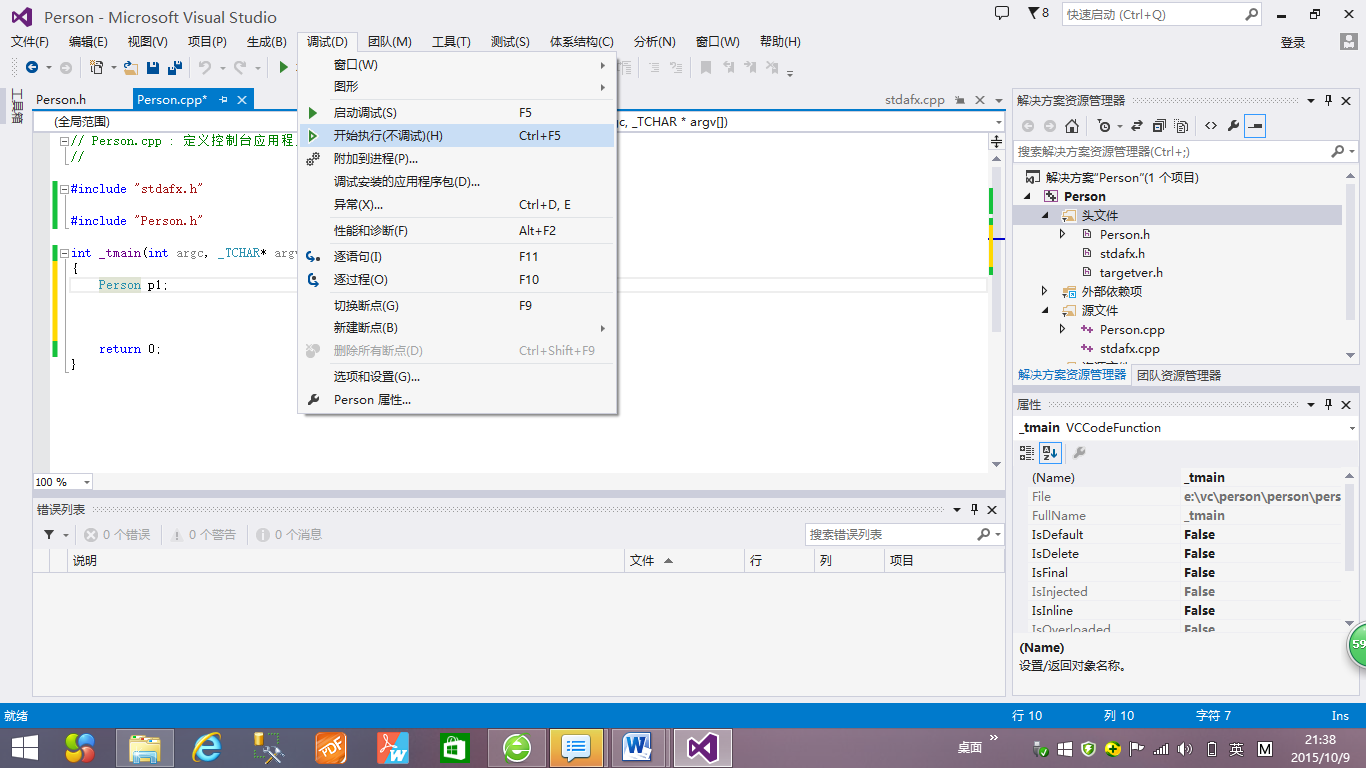
1. 按应用程序向导选择下列各项：控制台应用程序(Console application)，预编译头(Precompiled header)，安全开发声明周期(SDL)检查；
2. 添加头文件Person.h
3. 在头文件Person.h中定义Person类及其他各派生类，并实现各类的成员函数。



（5）在文件Person.cpp开头的地方引用头文件Person.h,编辑主程序，创建各派生类类对象，调用成员函数。

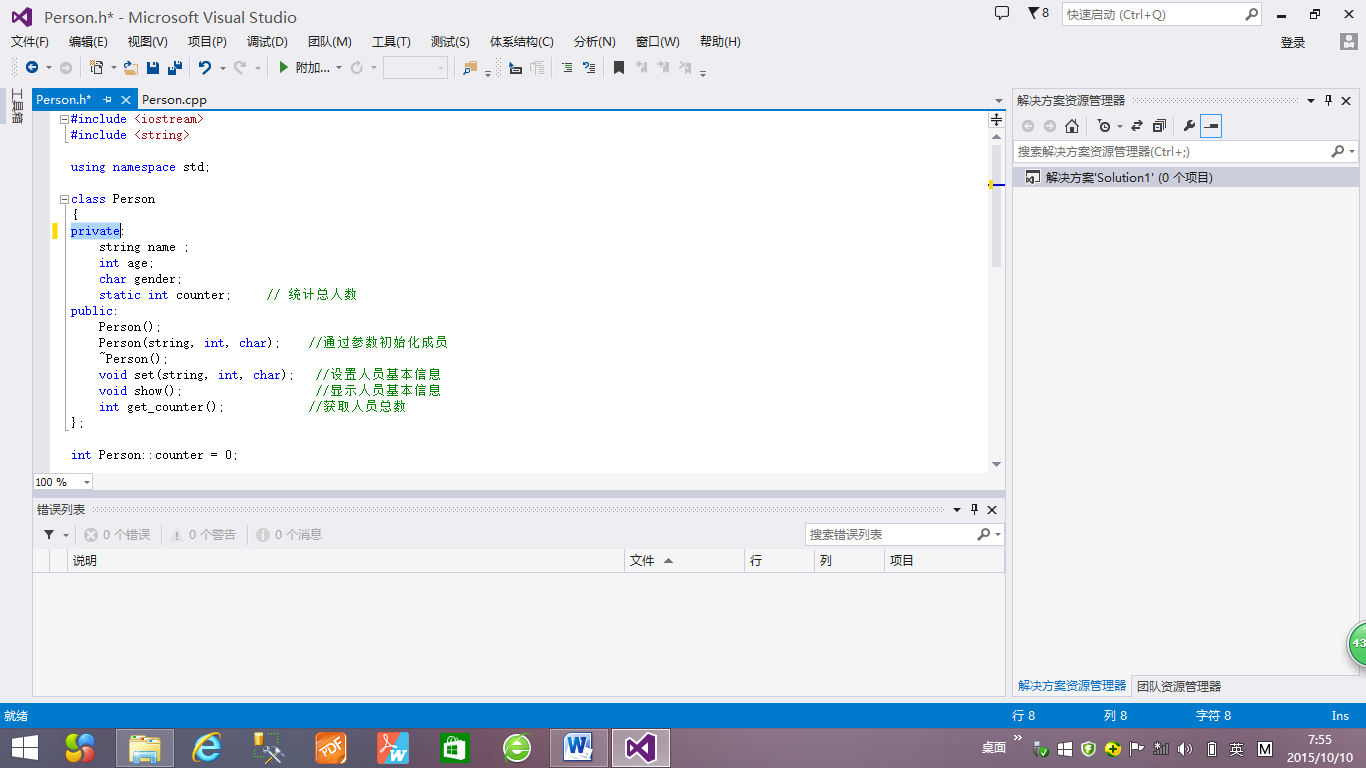


（6）选择菜单“调试”->开始执行(不调试)，运行程序，若有错误，需按错误提示逐个修改。

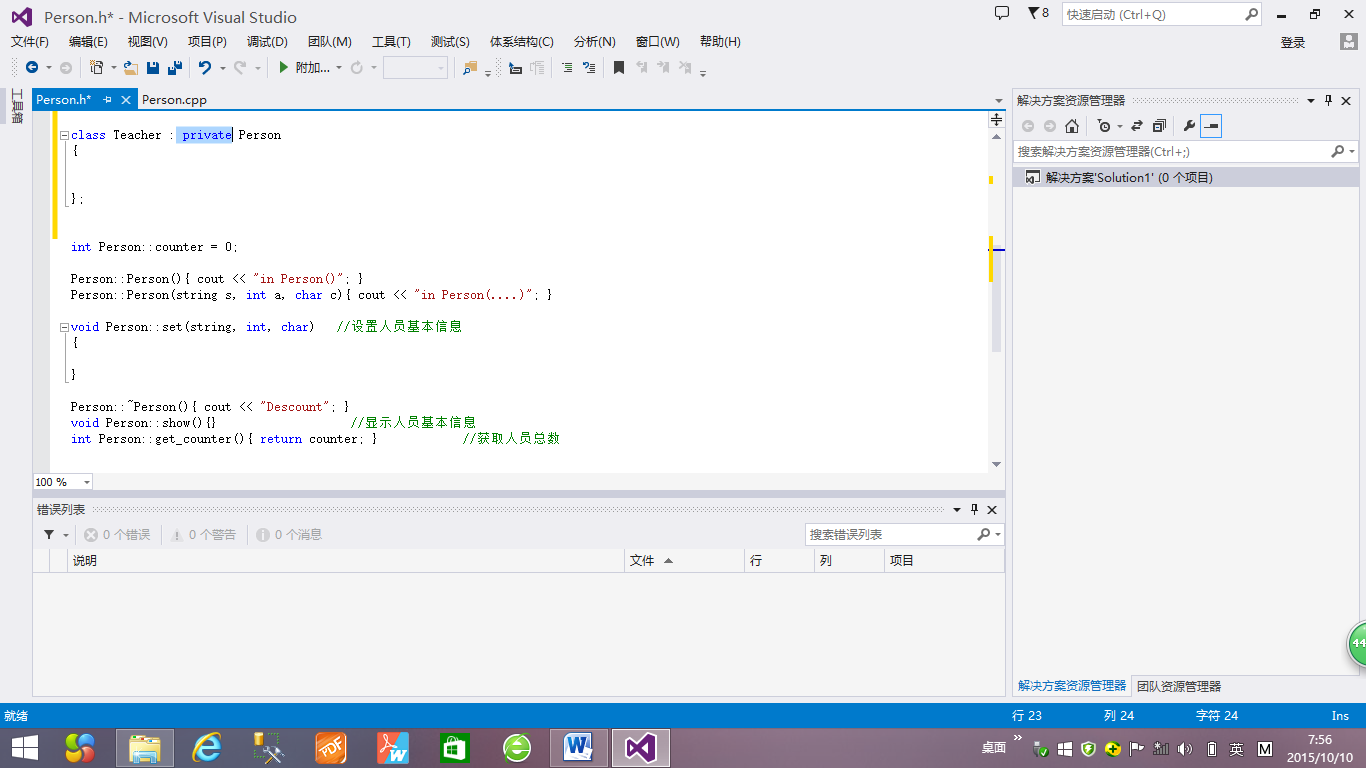


（7）运行程序，验证各类的功能。

（8）修改Person类数据成员的属性为private，重新编译程序，分析程序中出现错误的原因。

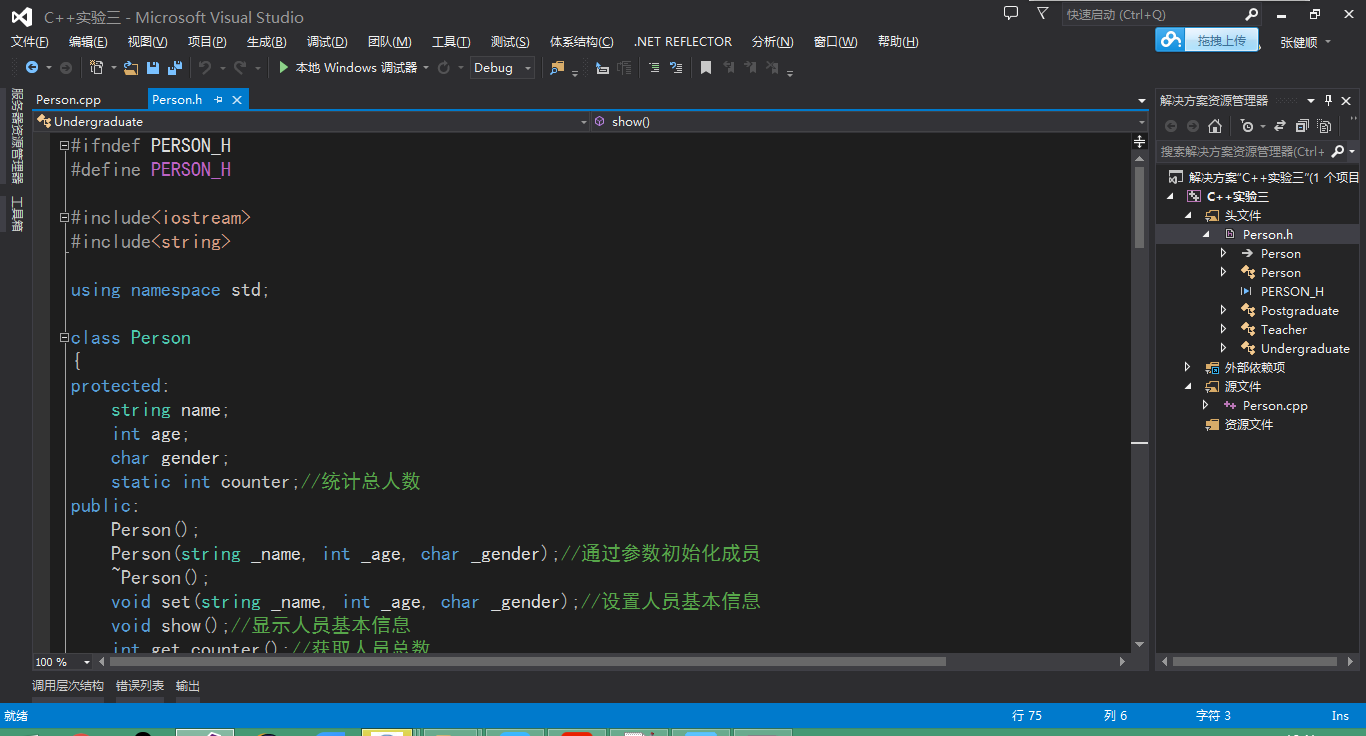


（9）将继承方式分别修改为private继承和Protected继承，运行程序，分析不同继承方式对基类成员在派生类中访问属性的影响。



1. **实验结果与分析（含重要数据结果分析或核心代码流程分析）**

（1）建立工程后，编写头文件Person.h，在头文件Person.h中定义Person类及其他各派生类，并实现各类的成员函数。



int Person::counter = 0;

Person::Person(){ ++counter; };

Person::Person(string \_name, int \_age, char \_gender)

:name(\_name), age(\_age), gender(\_gender){ Person(); }

Person::~Person(){};

void Person::set(string \_name, int \_age, char \_gender){

name = \_name;

age = \_age;

gender = \_gender;

}

void Person::show(){

cout << "name:" << name<< " age:" << age << " gender:" << gender<< endl;

}

int Person::get\_counter(){

return counter;

}

派生类以及各类的成员函数

A、Teacher类

class Teacher :public Person //公有继承

{

private:

int EmployeeID;

string title;

public:

Teacher(string \_name, int \_age, char \_gender, int \_ID = 01, string \_title = "professor")

:Person(\_name,\_age , \_gender), EmployeeID(\_ID), title(\_title){}

//构造函数需要

~Teacher(){}

void show(){

cout << "name:" << name << " age:" << age << " gender:" << gender<<

" EmployeeID:"<<EmployeeID<<" title:"<<title<< endl;

}

void set(string \_name, int \_age, char \_gender, int \_ID = 01, string \_title = "professor"){

Person::set(\_name, \_age, \_gender);

EmployeeID = \_ID; title = \_title;

}

};

B、Undergraduate类

class Undergraduate :public Person //公有继承

{

private:

int StuID;

string depart;

public:

Undergraduate(string \_name, int \_age, char \_gender, int \_ID, string \_depart)

:Person(\_name, \_age, \_gender), StuID(\_ID), depart(\_depart){}

~Undergraduate(){}

void show(){

cout << "name:" << name << " age:" << age << " gender:" << gender <<" StuID:" << StuID << " depart:" << depart<< endl;

}

void set(string \_name, int \_age, char \_gender, int \_ID , string \_depart){

Person::set(\_name, \_age, \_gender);   
 StuID = \_ID; depart = \_depart;

}

};

C、 Postgraduate类

class Postgraduate :public Undergraduate //公有继承

{

private:

string teacher;

public:

Postgraduate(string \_name, int \_age, char \_gender, int \_ID, string \_depart, string \_teacher)

:Undergraduate(\_name, \_age, \_gender, \_ID, \_depart), teacher(\_teacher){}

~Postgraduate(){};

void show(){

Undergraduate::show();

cout << "teacher:" << teacher << endl;

}

void set(string \_name, int \_age, char \_gender, int \_ID, string \_depart, string \_teacher){

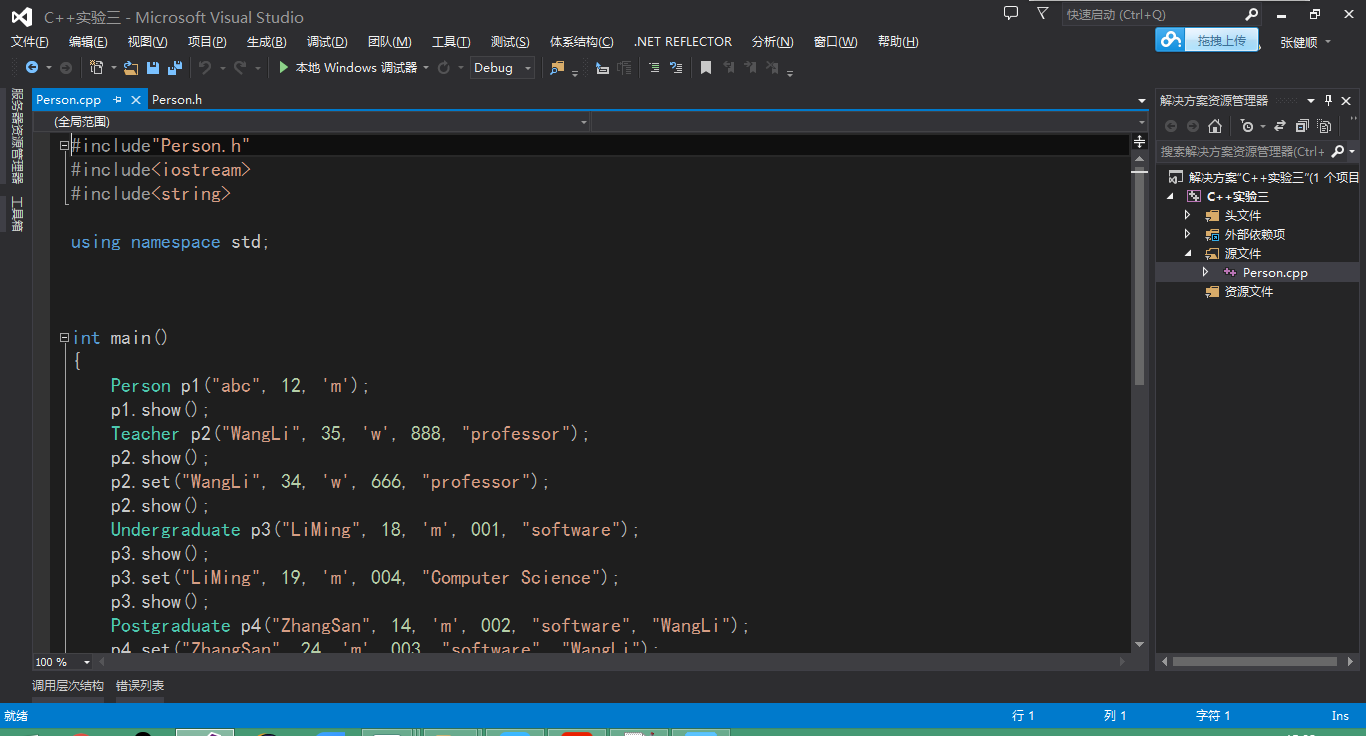
Undergraduate::set(\_name, \_age, \_gender, \_ID, \_depart);

teacher = \_teacher;

}

};

（2）在文件Person.cpp开头的地方引用头文件Person.h,编辑主程序，创建各派生类类对象，调用各自的构造函数，调用成员函数，调用相应的show()方法和set()方法。同时在最终通过调用get\_counter()方法，统计创造的对象个数，运行程序，验证各类的功能。



int main(){

Person p1("abc", 12, 'm');

p1.show();

Teacher p2("WangLi", 35, 'w', 888, "professor");

p2.show();

p2.set("WangLi", 34, 'w', 666, "professor");

p2.show();

Undergraduate p3("LiMing", 18, 'm', 001, "software");

p3.show();

p3.set("LiMing", 19, 'm', 004, "Computer Science");

p3.show();

Postgraduate p4("ZhangSan", 14, 'm', 002, "software", "WangLi");

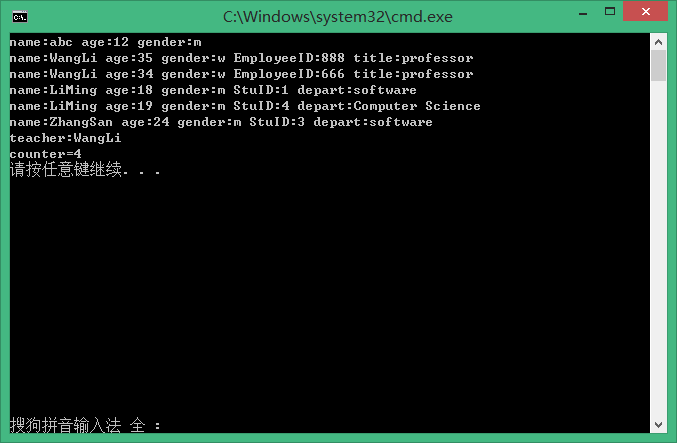
p4.set("ZhangSan", 24, 'm', 003, "software", "WangLi");

p4.show();

cout<<"counter="<<p1.get\_counter()<<endl;

return 0;

}

（3）修改Person类的数据成员属性为private，重新编译程序，分析错误原因。原属性为protected

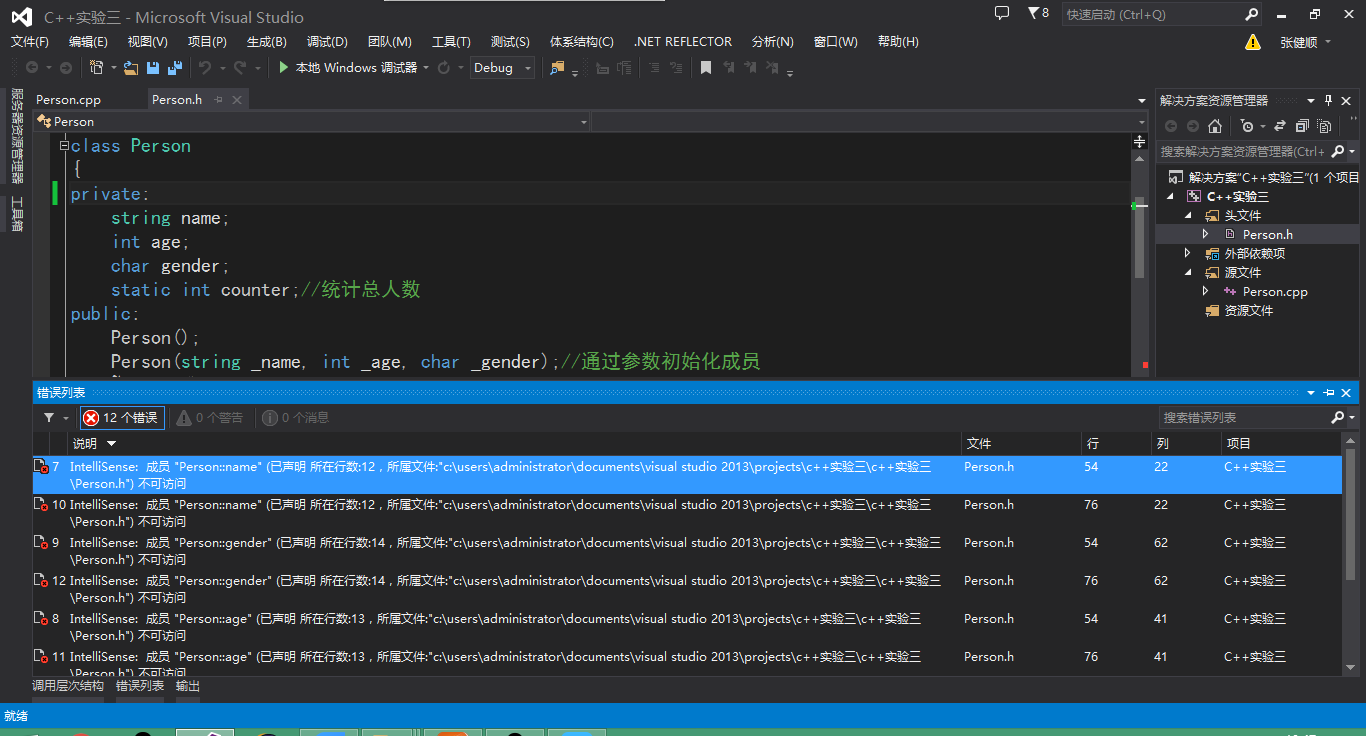
private:

string name;

int age;

char gender;

static int counter;//统计总人数



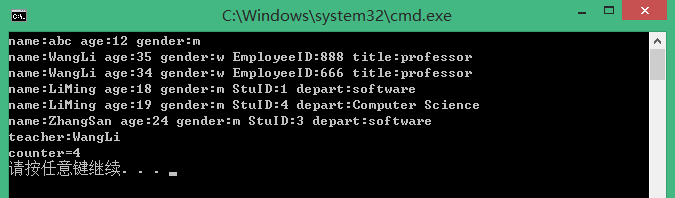
属性声明修改为private之后报错

（4）由于把父类也就是Person类成员属性为private后，这些成员属性在该类中仍然能够访问，但在其子类中如果直接访问成员属性，则无法进行访问，因为被声明为private之后，属性仅为本类所拥有，而子类无法继承父类中的属性。

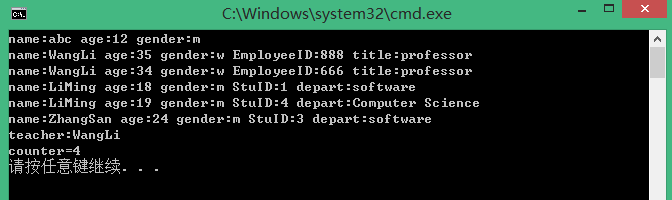
将继承方式分别修改为private继承和Protected继承，运行程序，分析不同继承方式对基类成员在派生类中访问属性的影响。

修改为private：

class Undergraduate :private Person

修改为protected：

class Teacher :protected Person



原成员属性为protected，原继承方式为public，现在将继承方式修改为private，protetced后，相应的继承的成员访问属性在子类中将变成private和protected。但不影响该程序的正常执行，所以程序执行结果与原结果相同。

1. **总结及心得体会：**

（1）派生类的成员函数不能直接访问基类的private成员，只能访问基类的public成员。如果要在派生类中使用基类的成员可以通过 基类名：：基类成员。

|  |  |  |  |
| --- | --- | --- | --- |
| 基类属性  访问控制 | public | protected | private |
| public | public | protected | private |
| protected | protected | protected | private |
| private | 不可访问 | 不可访问 | 不可访问 |

（2）派生类对象构造时，会先调用基类的构造函数，在调用派生类的构造函数，又由于堆栈的原因，所有释放内存时会先调用派生类对象的析构函数，在调用基类的析构函数。

（3）在派生类中要想调用基类的同名成员，应使用基类名限定。派生类构造函数的声明为：派生类构造函数（参数列表）：基类（参数列表），成员（参数列表），...

**八、对本实验过程及方法、手段的改进建议：**

（1）在构造函数和析构函数的函数体中加入一条输出语句，再在主函数中创建对象时，能够清楚明朗地看到派生类对象的创建过程与销毁过程；

（2）该实验中本科生（Undergraduate）类和研究生（Postgraduate）类拥有很多相同的属性成员，只是研究生（Postgraduate）类多了一个导师属性，可以让研究生类（Postgraduate）继承本科生类（Undergraduate），同时研究生类中的导师属性可以声明为老师（Teacher）类，从而使代码达到最大限度的重用。

**报告评分：**

**指导教师签字：**