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：16.10.25**

1. **实验名称：类和对象**
2. **实验学时：4**
3. **实验目的：**

1）理解类和对象的概念，掌握声明类和定义对象的方法；

2）掌握构造函数和析构函数的实现方法；

3）初步掌握使用类和对象编制C++程序；

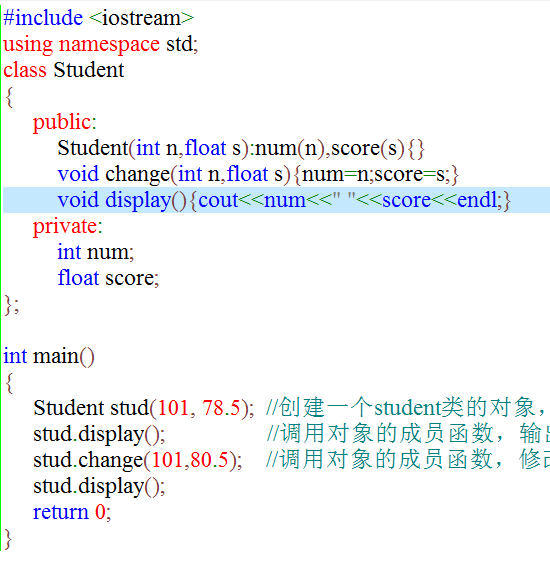
4）掌握使用对象、对象指针和对象引用作为函数参数的方法；

5）掌握静态数据成员和静态成员函数的使用方法；

6）理解友元的概念和掌握友元函数的使用方法。

1. **实验内容：**

1）阅读一段代码，分析其执行过程，然后上机运行，对比输出结果；修改上面的程序，增加一个fun函数，改写main函数。在main函数中调用fun函数，在fun函数中调用change和display函数。在fun函数中使用对象的引用（Student&）作为形参。



2）商店销售某一商品，商店每天公布统一的折扣（discount）。同时允许销售人员在销售时灵活掌握售价（price），在此基础上，对一次购10件以上者还可以享受9.8折优惠。现已知当天3个销货员销售情况为：

销货员号（num） 销货件数（quantity） 销货单价（price）

101 5 23.5

102 12 24.56

103 100 21.5

请编些程序，计算出当日此商品的总销售款sum以及每件商品的平均售价。要求用静态数据成员和静态成员函数。

3）设计一个立方体类Box，它能计算并输出立方体的体积和表面积。要求写出Box类的构造函数和析构函数。

4）阅读一段代码，将程序中的display函数不放在Time类中，而作为类外的普通函数，然后分别在Time和Date类中将display声明为友元函数。在主函数中调用display函数，display函数分别引用Time和Date两个类的对象的私有数据输出 年，月，日和时，分，秒。修改后上机调试和运行。

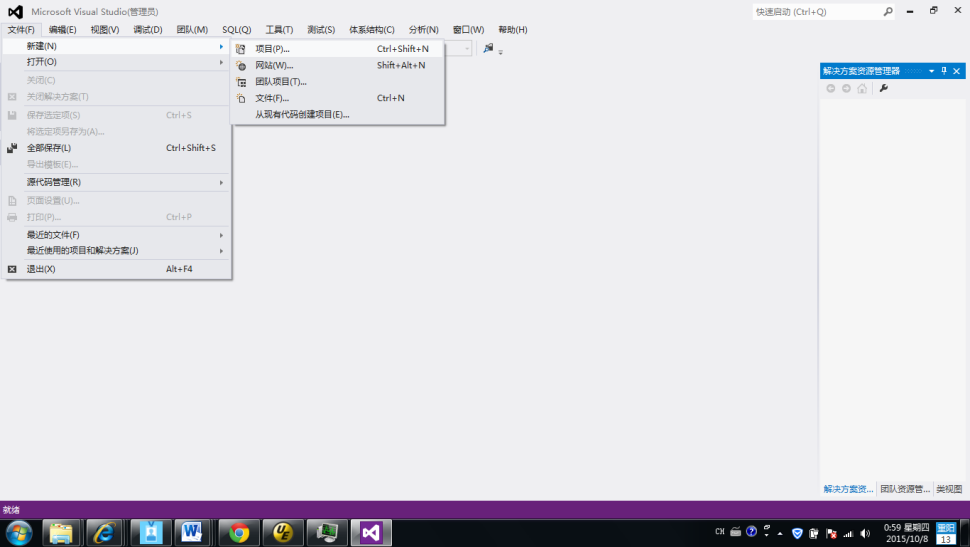
|  |
| --- |
| #include<iostream>  using namespace std;  class Date; //对Date类的提前引用声明  class Time //定义Time类  {  public:  Time(int, int, int); //构造函数  void display(Date &); //display是成员函数，形参是Date类对象的引用  private:  int hour;  int minute;  int sec;  };  class Date //声明Date类  {  public:  Date(int, int, int);  friend void Time::display(Date &); //声明Time中的display函数为友元成员函数  private:  int month;  int day;  int year;  };  Time::Time(int h, int m, int s) //类Time的构造函数  {  hour=h;  minute=m;  sec=s;  }  void Time::display(Date &d) //display的作用是输出年，月，日和时，分，秒  {  cout<<d.month<<"/"<<d.day<<"/"<<d.year<<endl;  cout<<hour<<":"<<minute<<":"<<sec<<endl;  }  Date::Date(int m,int d,int y) //类Date的构造函数  {  month=m;  day=d;  year=y;  }  int main()  {  Time t1(10,13,56); //定义Time类对象t1  Date d1(12,25,2004); //定义Date类对象d1  t1.display(d1); //调用t1中的display函数，实参是Date类对象d1  return 0;  } |

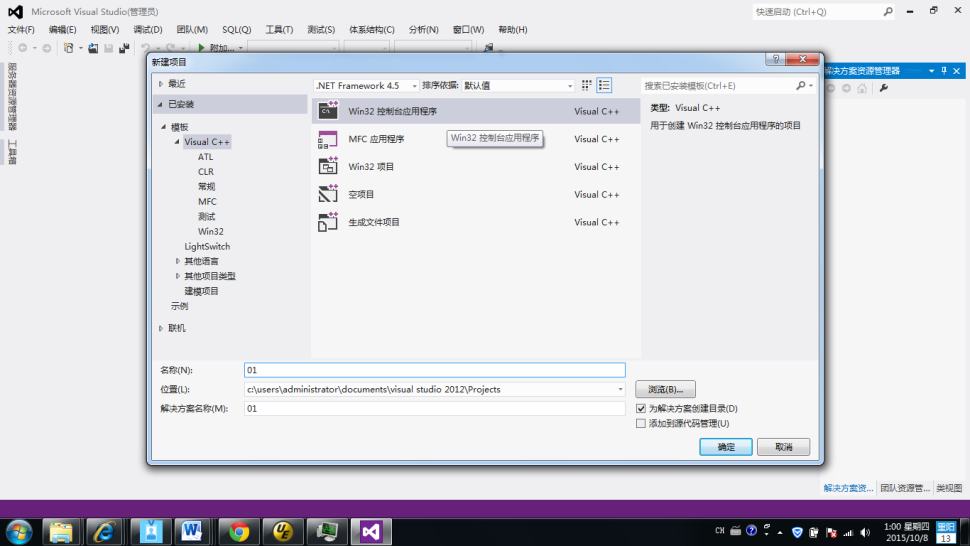
1. **实验器材（设备、元器件）：**

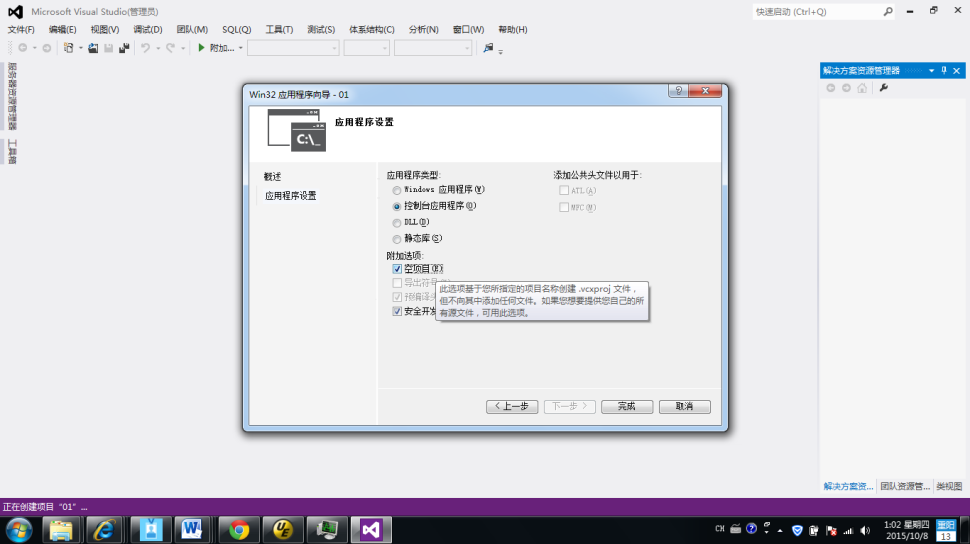
PC机一台，Windows 8.1，Visual Studio 2013

1. **实验步骤：**

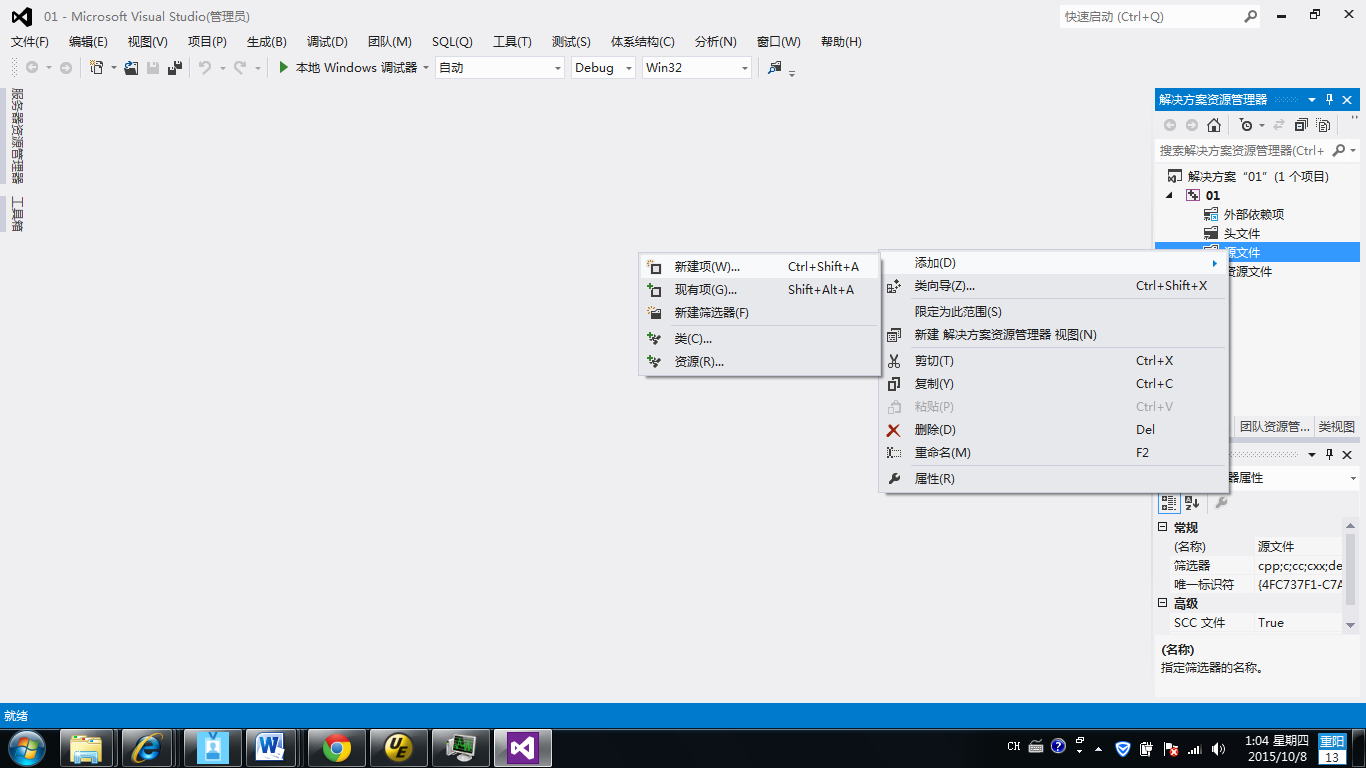
1、创建一个新的工程项目，选择项目的类型为“win32控制台应用程序”，主要选项如图所示。



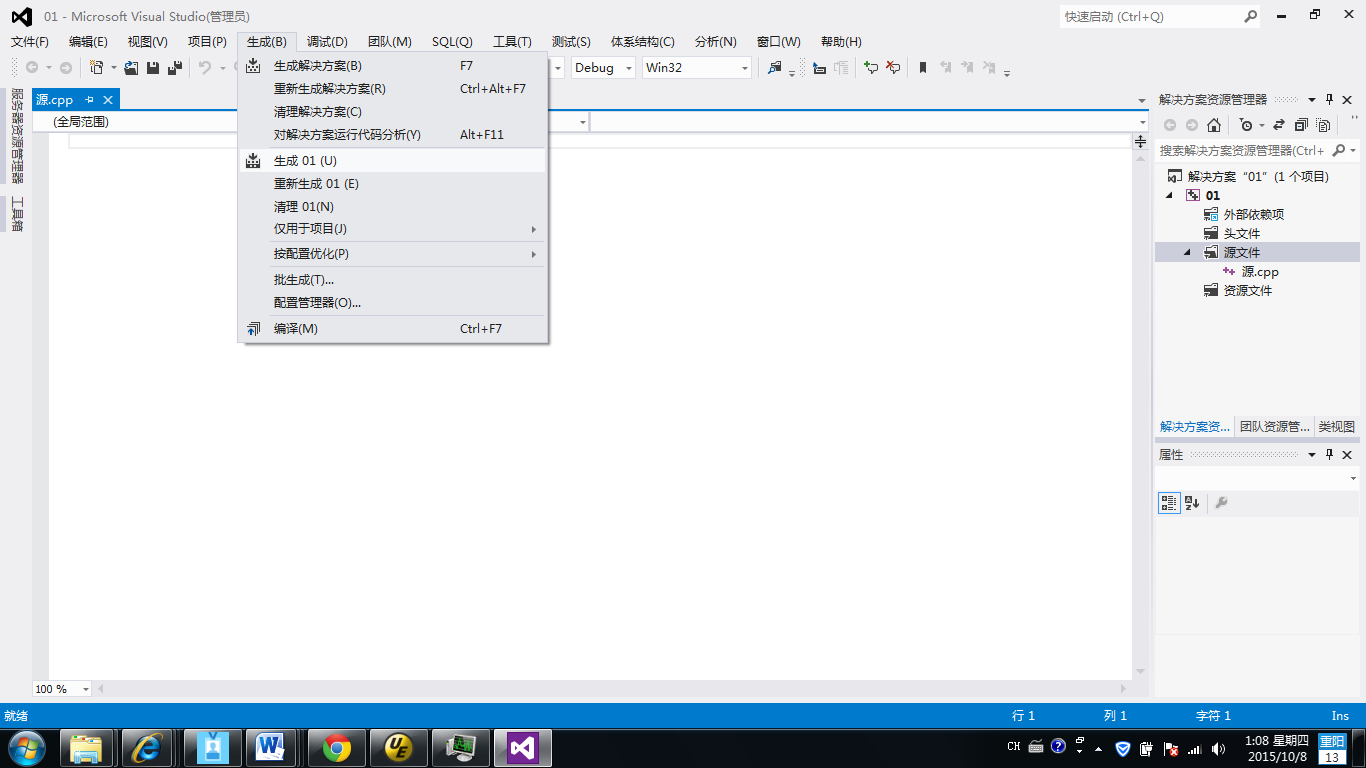


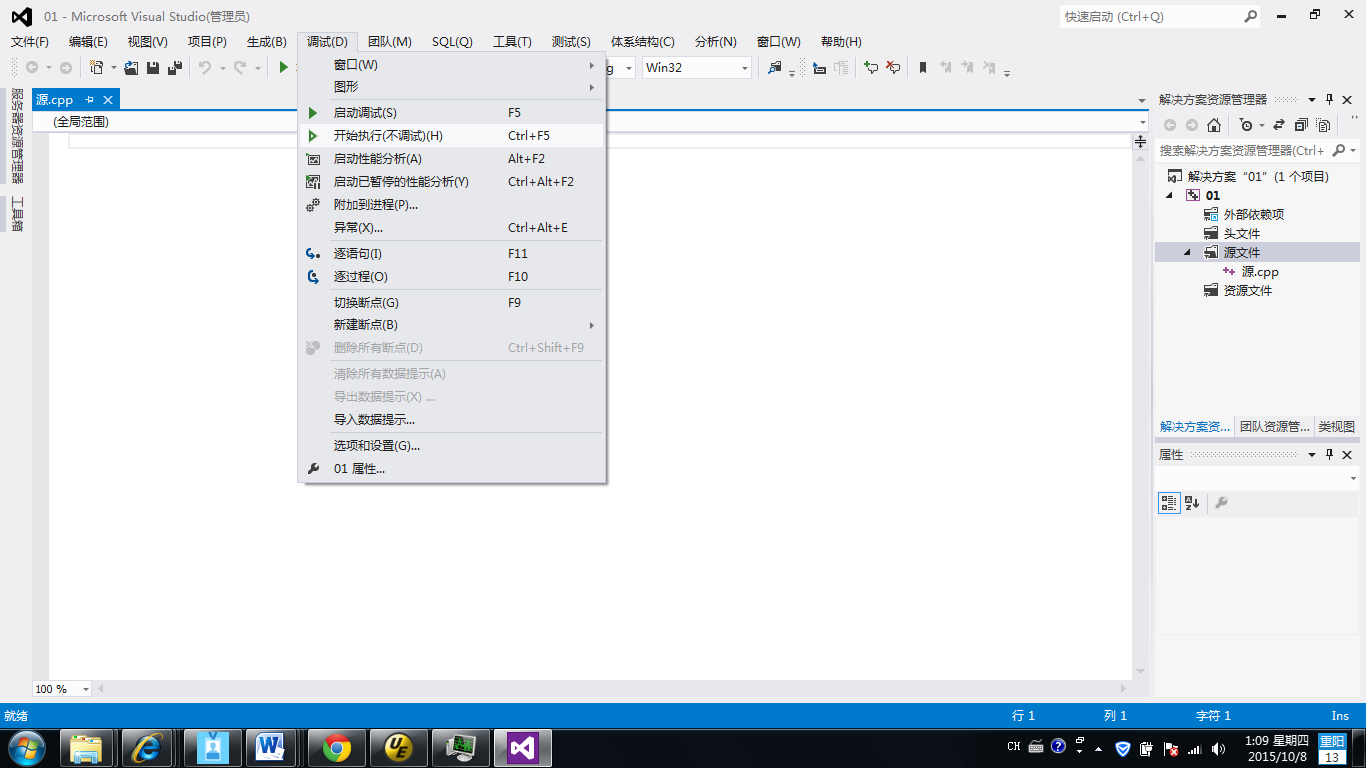


2、在创建完项目后，在项目的源文件中点击右键-》选择添加“新建项”-》选择“C++文件”，即可在新文件中输入源代码。



3、输入完源代码后，需要编译程序及运行程序。





1. 按照实验内容逐步完成实验操作
2. **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）**

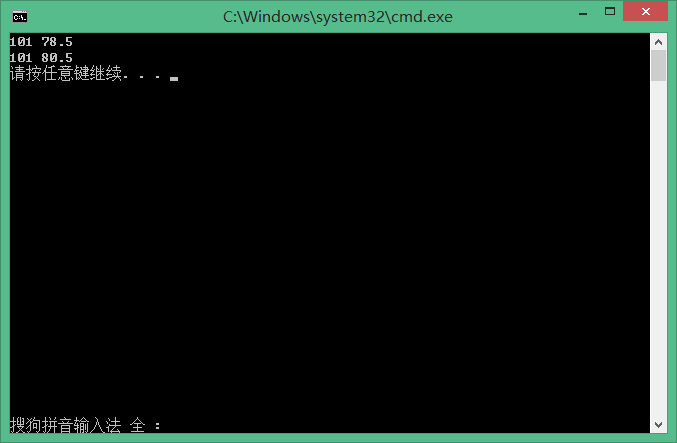
1. 原题目所给代码执行过程：

（1）创建一个Student对象stud并调用类中定义的构造函数进行初始化；

（2）调用对象的成员函数display()来输出初始化对象后的属性值；

（3）调用对象的成员函数change()来对对象的属性值进行修改；

（4）再次调用对象的成员函数display()来输出修改后的对象的属性值。



题目所给代码的两次输出（初始化、修改后）

修改程序：增加一个fun函数，改写main函数。在main函数中调用fun函数，在fun函数中调用change和display函数。在fun函数中使用对象的引用（Student&）作为形参。

#include<iostream>

using namespace std;

class Student{

public:

Student(int n, float s) :num(n), score(s){}

void change(int n, float s){

num = n; score = s;

}

void display(){

cout <<"修改代码后的输出结果："<< num << " " << score << endl;

}

private:

int num;

float score;

};

void fun(Student& stu) //传入student引用

{

stu.change(101, 80.5);

stu.display();

}

int main()

{

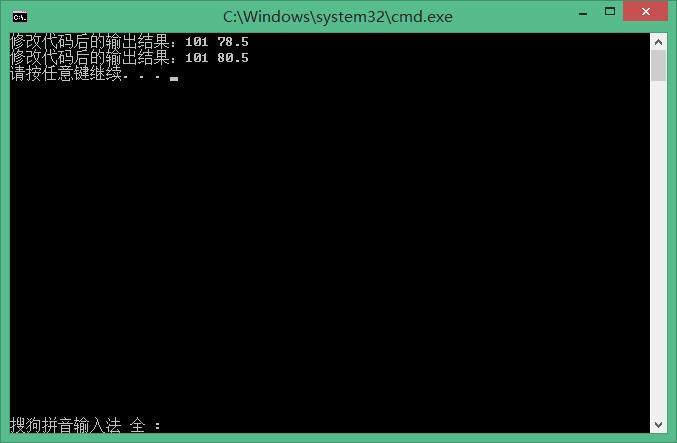
Student stud(101, 78.5);

stud.display();

fun(stud);

return 0;

}

****

修改代码后的输出结果

2.解决问题过程：

（1）设计出SalesMan类（对应的有销货员号、销货件数以及销货单价的成员变量同时设计出相应的统计方法）；

（2）根据题目要求使用静态数据成员和静态成员函数从而达到各个对象共享成员变量以及成员方法的效果，同时可自定义折扣：

static void dispaly();

double SalesMan::discount = 0.05;

double SalesMan::sumSalePrice = 0;

int SalesMan::sumSaleQuantity = 0;

1. SalesMan中相应方法，源码如下：

class SalesMan{

public:

SalesMan(){}

SalesMan(int num, int quantity, double price){

singleQuantity = quantity;

if (quantity>10){singlePrice = quantity\*price\*0.98;} else {singlePrice= quantity\*price;}

singlePrice \*= 0.95;

}

void total(){

sumSalePrice += singlePrice;

sumSaleQuantity += singleQuantity;

}

static void display(){

cout << "销售总额：" << sumSalePrice << endl << "平均售价："

<< sumSalePrice / sumSaleQuantity<<endl;

}

private:

int singleQuantity;

double singlePrice;

static int sumSaleQuantity;

static double discount;

static double sumSalePrice;

};

1. 控制台输入在main函数中对应的实现：

int main(){

SalesMan S[10]; int m，num，quantity; double price;

cout << "售货员人数：" ; cin >> m;

cout << "销货员号 销货件数 销货单价" << endl;

for (int i = 0; i < m; i++){

cin >> num >> quantity >> price;

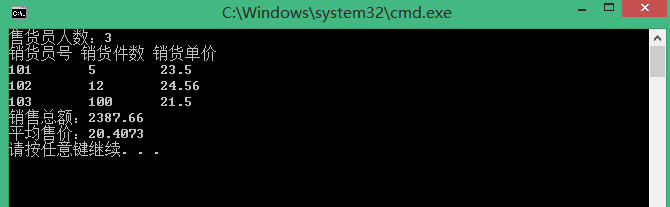
SalesMan temp(num, quantity, price); S[i] = temp;}

for (i = 0; i < m; i++) S[i].total();

SalesMan::display();

return 0;

}



输出结果

3.设计出的Box类（将其初始化为Box box(1,2,3);并在main函数中输出相应属性值以及调用求体积和求表面积的方法后的输出）

class Box{

public:

Box(int l, int w, int h) :length(l), width(w), height(h){

cout << "调用了构造函数！" << endl;

}

~Box(){

cout << "调用了析构函数"<< endl;

}

int Volume(){return height\*length\*width;}

int Superficialarea(){

return 2 \* (height\*width + height\*length + length\*width);

}

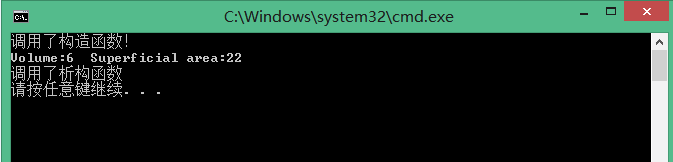
private:

int length;

int width;

int height;

};



输出结果

4.将程序中的display函数不放在Time类中，而作为类外的普通函数，然后分别在Time和Date类中将display声明为友元函数

void display(Date &d,Time &t){

cout << d.month << "/" << d.day << "/" << d.year << endl;

cout << t.hour << ":" << t.minute << ":" << t.sec << endl;

}

class Time{

public:

Time(int, int, int); //构造函数类外声明

friend void display(Date &,Time &); //形参是Date类对象的引用

private:

int hour; int minute; int sec;

};

class Date{

public:

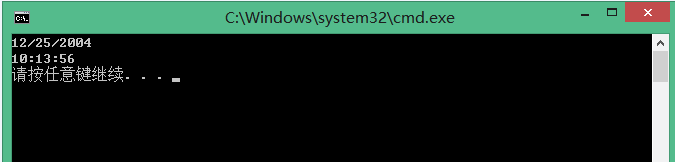
Date(int, int, int);

friend void display(Date &,Time &); //display函数为友元成员函数

private:

int month; int day; int year;

};



输出结果

1. **总结及心得体会：**

（1）根据需要解决的问题，确定好数据模型，并声明以及定义该数据模型对应的属性以及相应的方法从而定义出类，而对象则是类的具体定义与实现；

（2）构造函数主要用于对象的定义以及初始化的场景下，而复制构造函数则主要应用于对一个相同类型的对象的复制过程中以及某些函数中需要传递对象作为参数时需要调用复制构造函数；

（3）静态数据成员则主要应用于同类型多个对象共享一个数据成员，且均能对其进行读写；静态数据成员属于类，被该类的素有对象共享；

（4）需要在外部频繁访问类的某些私有成员，或者其他类需要和本类共享一个函数时则需要将函数或者某些类声明为友元函数与友元类。

（5）函数参数中传递引用和传递值/指针是完全不同的参数结合方式；前者传递给函数的是对象本身；而后两者传递的是对象的副本，或者是对象的地址，一旦涉及到对对象的修改时，操作的实体分别是对象本身和对象的复制品，从而使得结果会有所不同。

（6）对类中成员变量以及函数的定义可以在类外进行，但需要加上类名以及”::”进行限定。

（7）静态数据成员只能初始化一次，不能再次初始化，但可以对其做修改。

**九、对本实验过程及方法、手段的改进建议：**

（1）每个程序中涉及到的类的声明以及定义都可以放在头文件（.h文件）中，再在主程序中（.cpp文件）中引入该头文件，便于代码的管理；

（2）对于类的构造函数可以使用默认参数，防止初始化过程中遗漏某些参数，同时也能在主程序中无参地进行初始化，利于数据封装；

（3）构造函数和析构函数中可以输出相关函数执行语句，从而便于理解对象的产生与销毁的过程；

（4）对于某些需要用户输入大量数据的程序（例如第二题中的超市售价管理系统），可以考虑使用文件操作，利用文件的读写来进行信息的录入与写出。

**报告评分：**

**指导教师签字**

**：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：16.11.01**

1. **实验名称：C++的函数及运算符重载程序设计**
2. **实验学时：4**
3. **实验目的：**

通过实验练习，让学生理解类的成员函数重载的含义，理解构造函数的重载的意义，并掌握如何对构造函数重载；掌握函数重载的两种方式，分别通过成员函数和友元函数实现重载，理解两种实现方式的区别，最后，理解和掌握几种常用运算符重载的实现方法。

1. **实验内容：**

设计复数类MyComplex，复数的实部和虚部均为double类型，并且放在private段；重载复数类的构造函数，及复制构造函数；重载复数运算+，-，\*，/；重载复数流运算符<<,>>。同时实现返回复数实部、虚部、模的成员函数。编程设计实现下列函数：

myComplex();

myComplex(int a);

myComplex(int a,int b)

myComplex( myComplex& v)

double getReal(); //返回复数的实部

double getImaginary (); //返回复数的虚部

double getModulus (); //返回复数的模

myComplex& operator=( myComplex& rhs); //类对象的赋值

myComplex& operator+=( myComplex& rhs);

myComplex& operator-=( myComplex& rhs);

myComplex& operator\*=( myComplex& rhs);

myComplex& operator/=( myComplex& rhs);

friend myComplex operator+(myComplex m,myComplex n)

friend myComplex operator-(myComplex m,myComplex n)

friend myComplex operator\*(myComplex m,myComplex n)

friend myComplex operator/(myComplex m,myComplex n)

friend ostream& operator<<(ostream& os,myComplex c);

friend istream& operator>>(istream& is,myComplex& c);

要求：

（1）理解类的成员函数重载的含义；

（2）掌握类构造函数的重载定义与使用;

（3）掌握运算符的重载定义与使用；

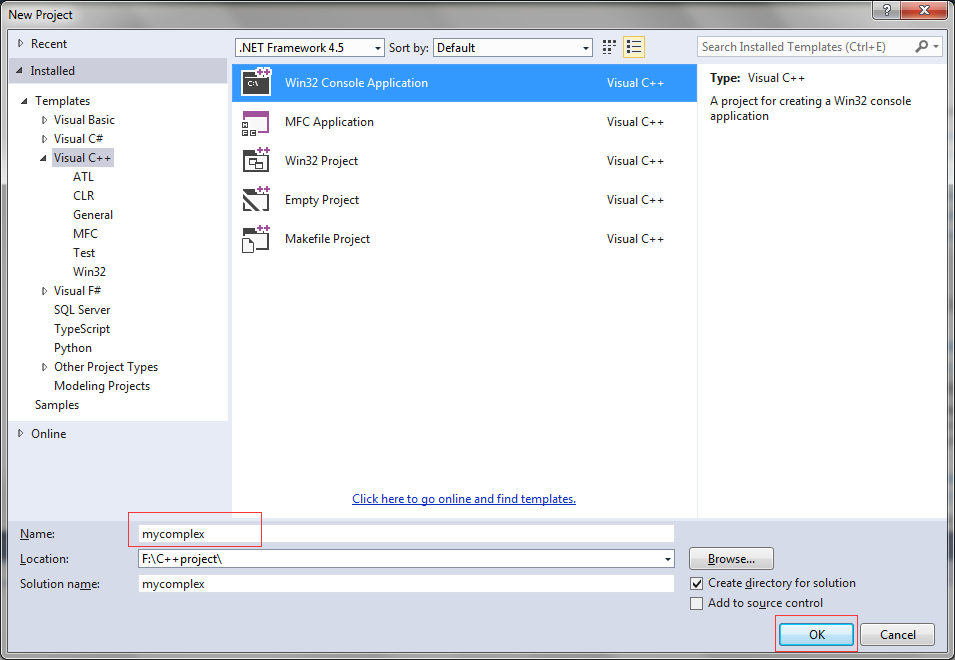
（4）掌握使用友元方法实现重载。

1. **实验器材（设备、元器件）：**

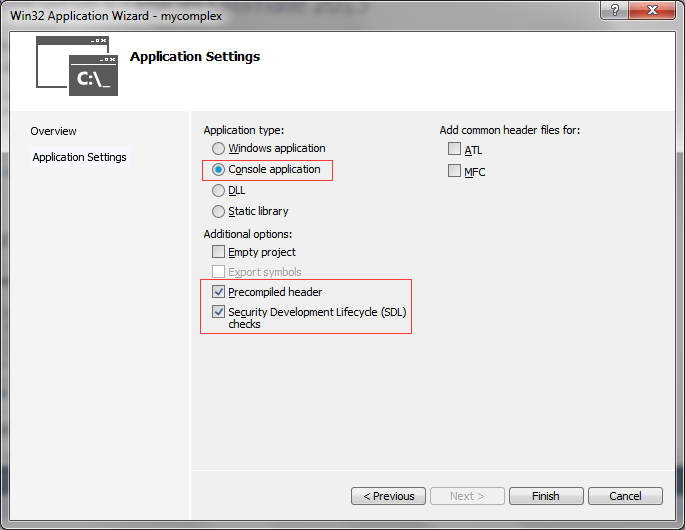
PC机一台，Windows 8.1，Visual Studio 2013

1. **实验步骤：**

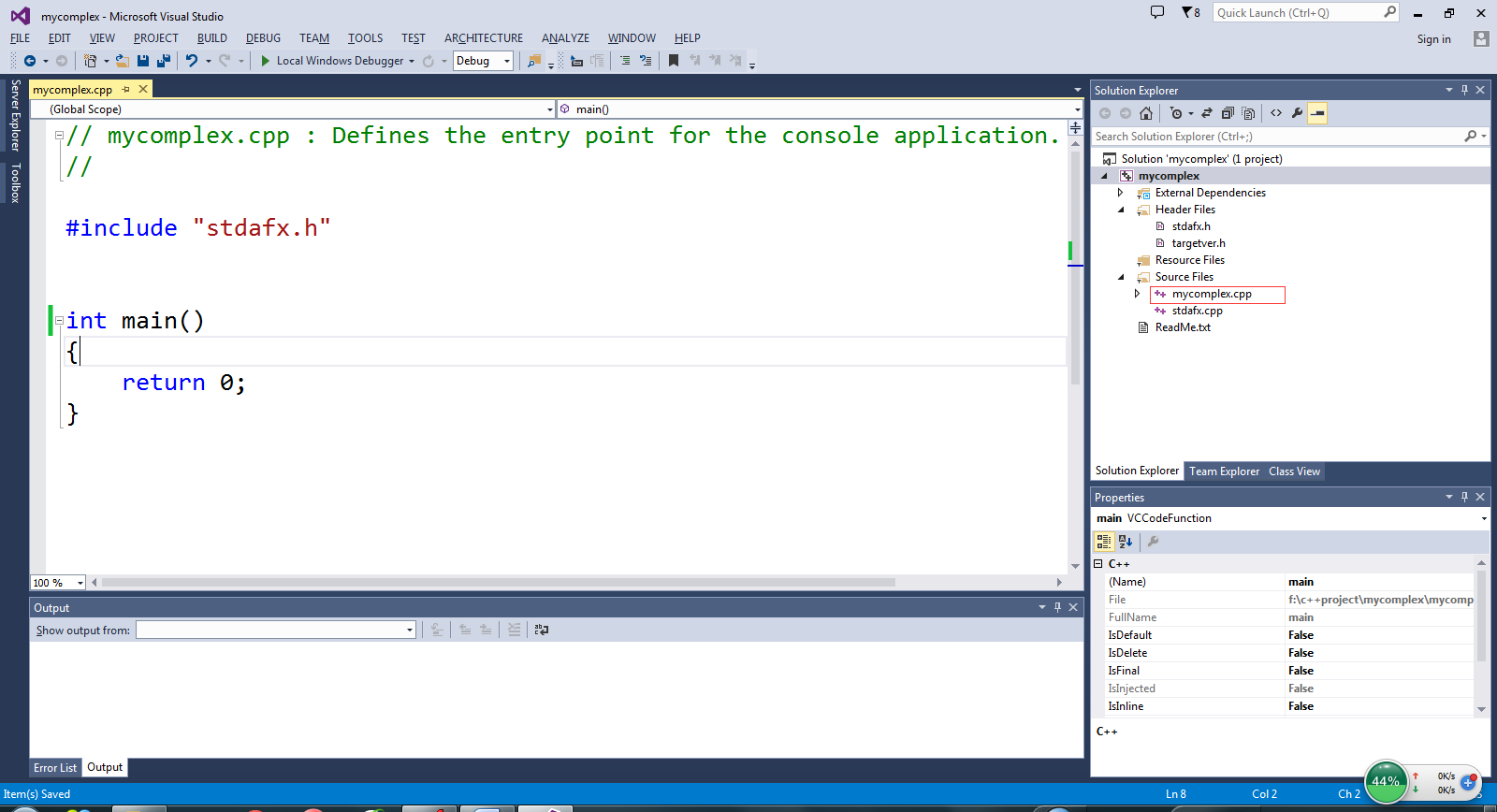
(1) 建立工程mycomplex



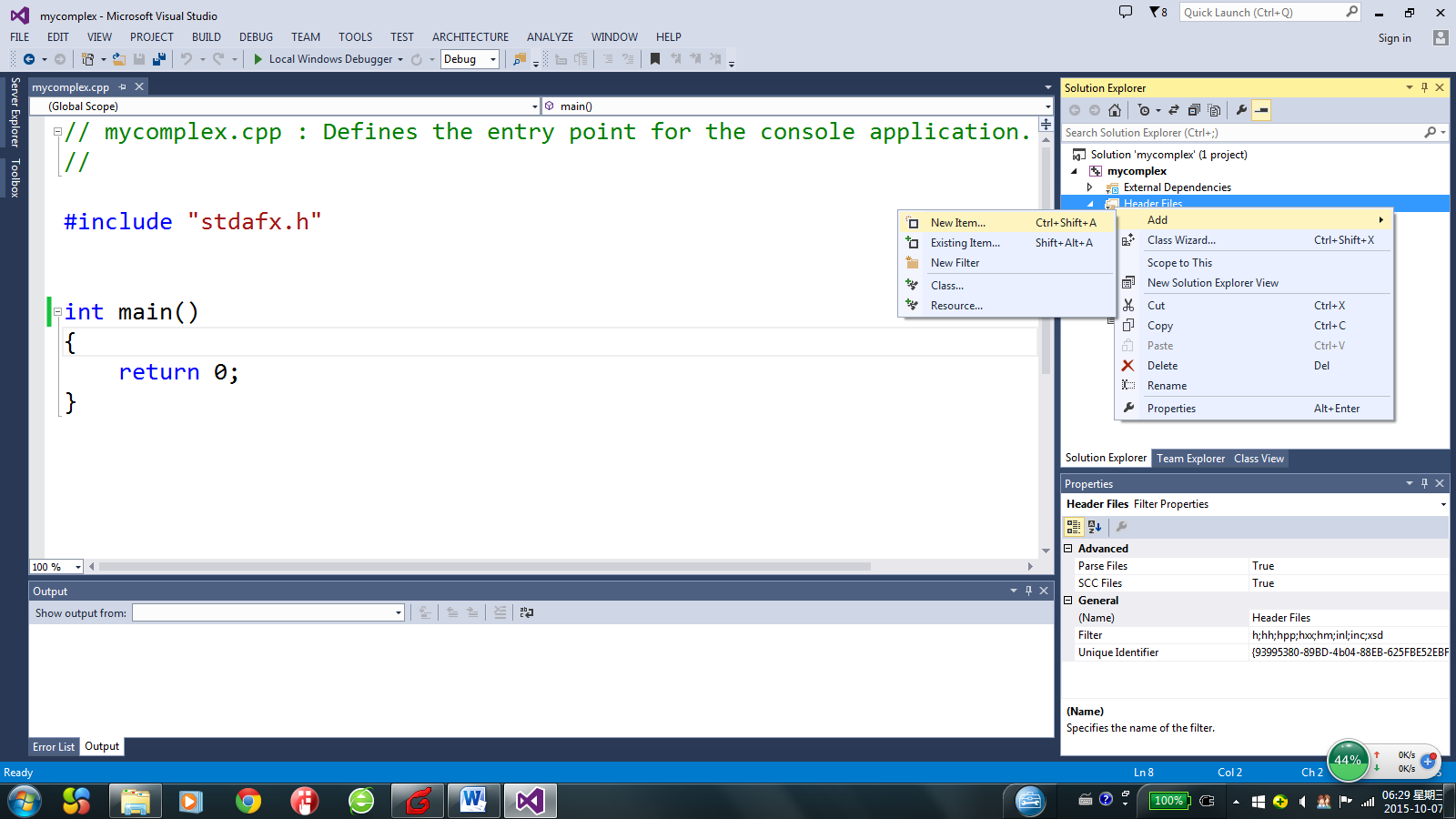
(2) 选择Console application，Precompiled header，SDL

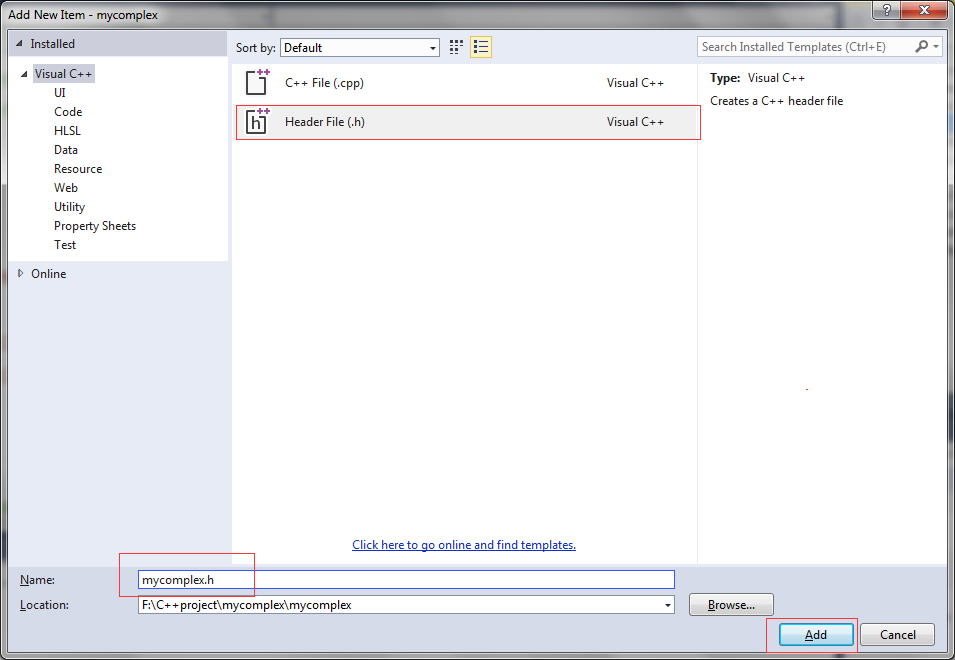


（3）生成如下所示mycomplex.cpp, 并将主程序入口改为 int main（）

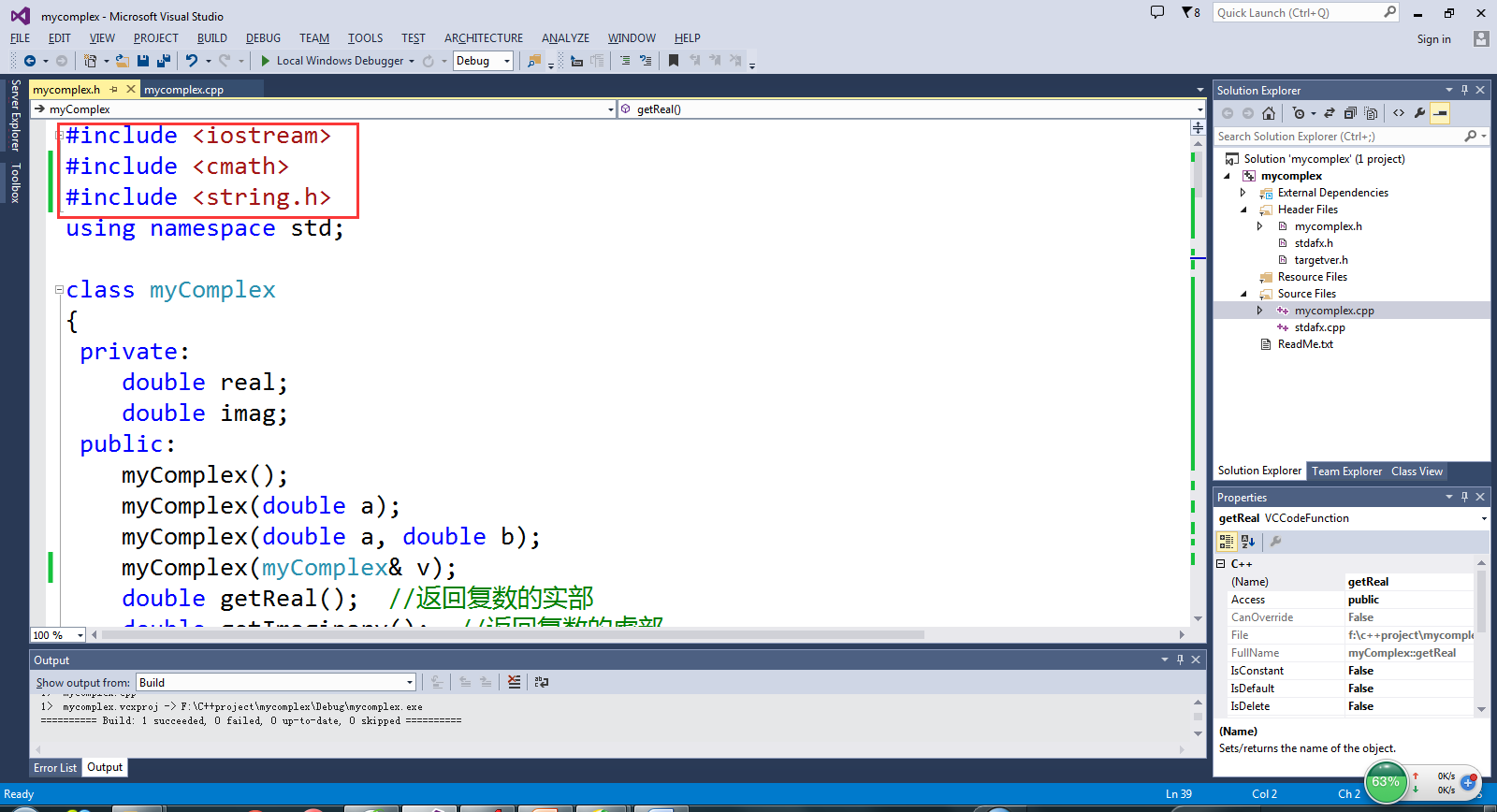


（4）参照下列图例，创建header文件mycomplex.h

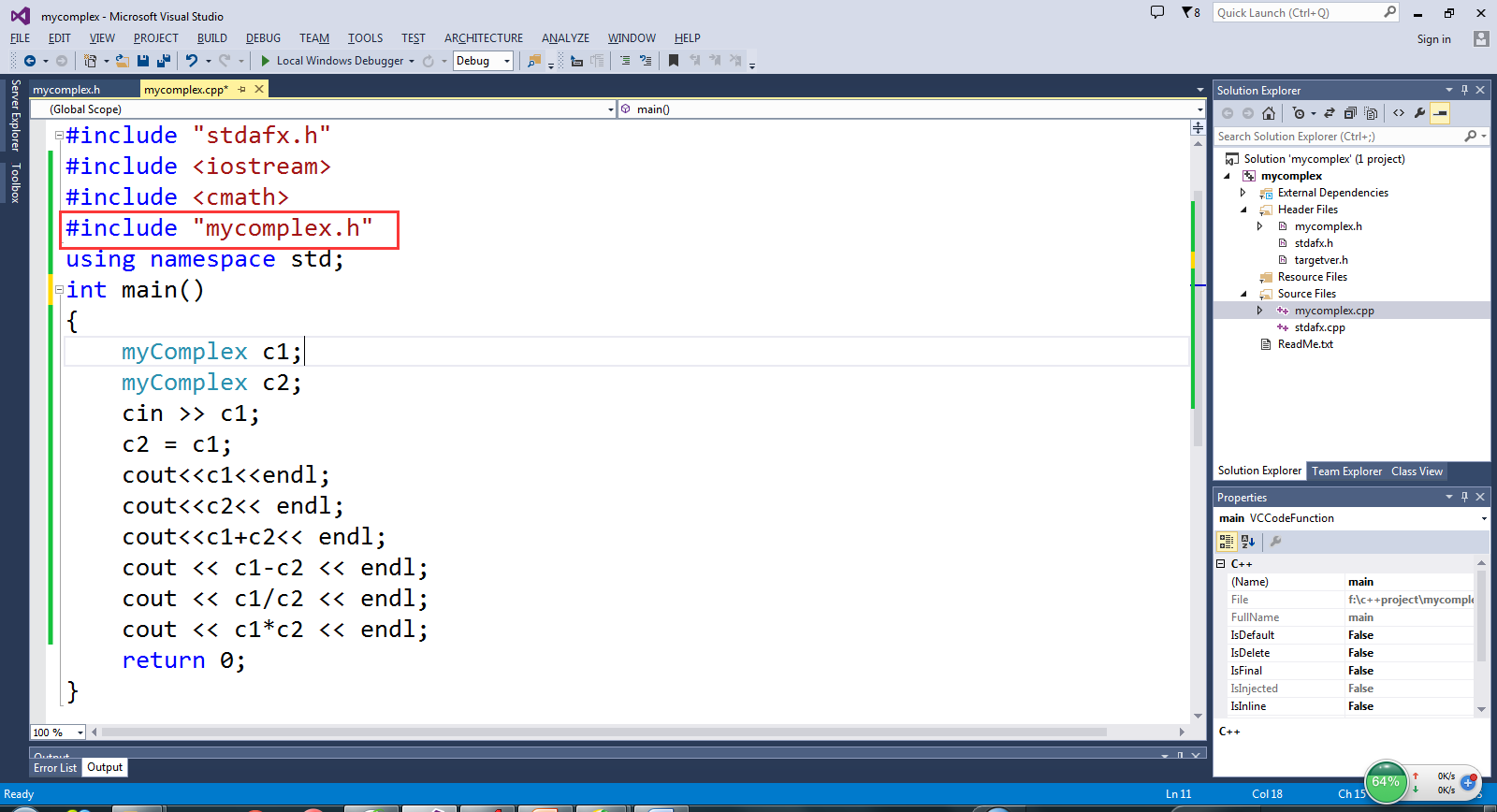




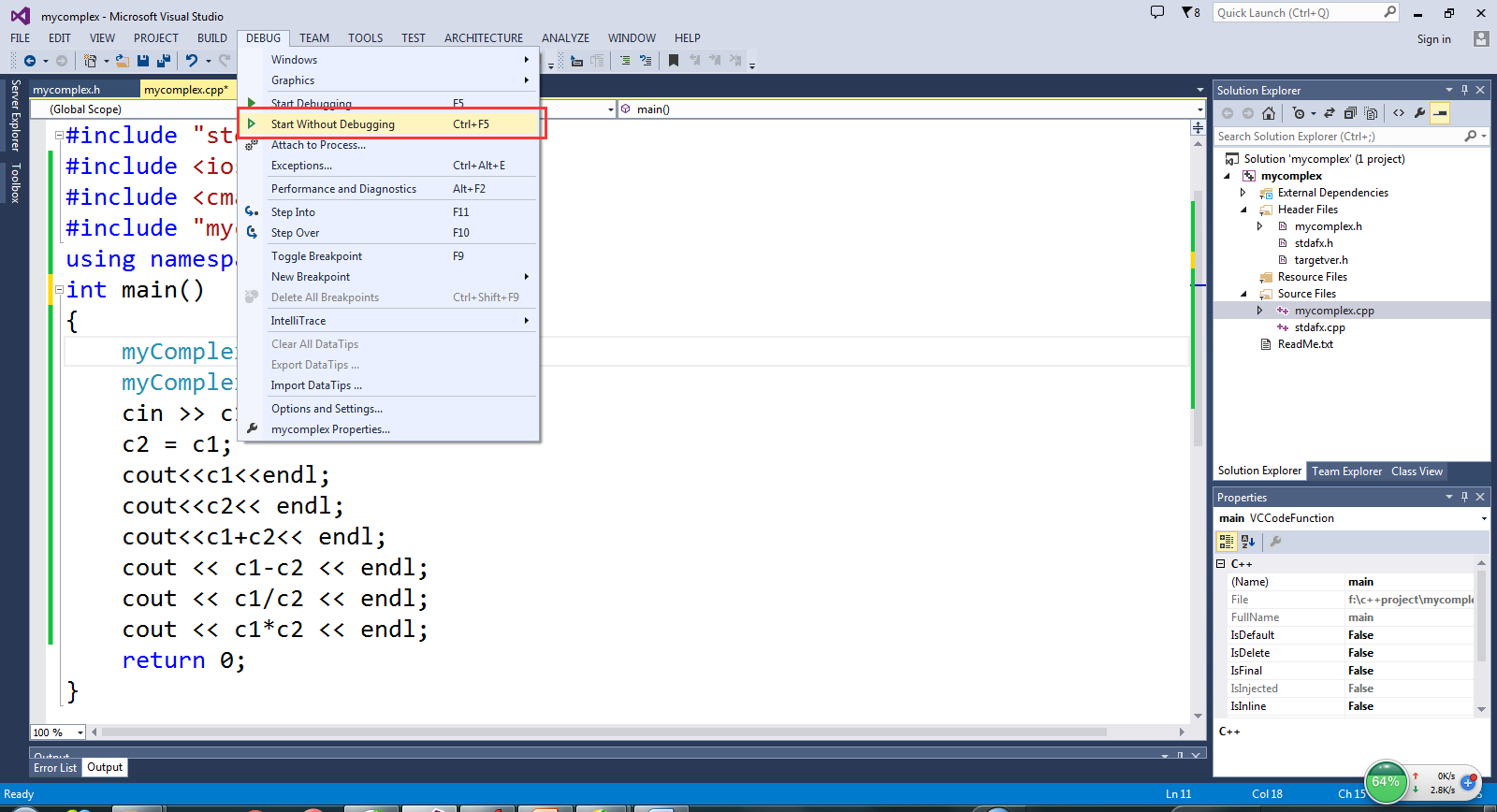
（5）在头文件中创建myComplex类，并实现类的构造函数、复制构造函数、成员函数、运算符重载的成员函数、友元重载的成员函数。



（6）在myComplex.cpp编辑主程序，验证重载后的运算符号。



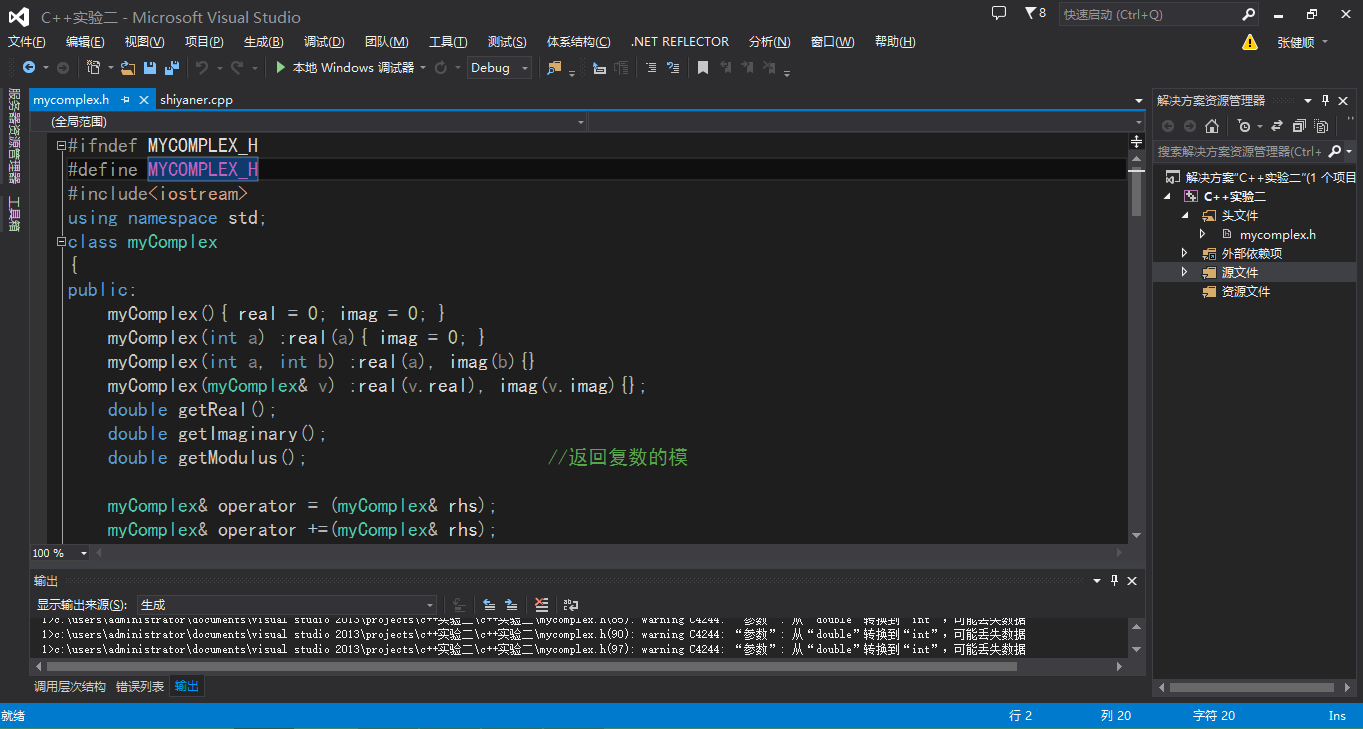
（7）调试运行程序



（8）运行程序，输入复数的实部和虚部后，显示结果

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）**
2. 建立工程后编写头文件，在头文件中创建myComplex类，并实现类的构造函数、复制构造函数、成员函数、运算符重载的成员函数、友元重载的成员函数；

其中类的构造函数可以考虑多种情形，参数的有无以及参数的个数，同时考虑运算符重载的成员函数以及友元重载的成员函数的区别。



class myComplex{

public:

myComplex(){ real = 0; imag = 0; }

myComplex(int a) :real(a){ imag = 0; }

myComplex(int a, int b) :real(a), imag(b){}

myComplex(myComplex& v) :real(v.real), imag(v.imag){};

double getReal(); //返回实部

double getImaginary(); //返回虚部

double getModulus(); //返回复数的模

myComplex& operator = (myComplex& rhs);

myComplex& operator +=(myComplex& rhs);

myComplex& operator -=(myComplex& rhs);

myComplex& operator \*=(myComplex& rhs);

myComplex& operator /=(myComplex& rhs);

friend myComplex operator + (myComplex& rhs1, myComplex& rhs2);

friend myComplex operator - (myComplex& rhs1, myComplex& rhs2);

friend myComplex operator \* (myComplex& rhs1, myComplex& rhs2);

friend myComplex operator / (myComplex& rhs1, myComplex& rhs2);

friend ostream& operator <<(ostream& os, myComplex &rhs);

friend istream& operator >>(istream& is, myComplex &rhs);

private:

double real,imag;

};

在类外定义类中声明的运算符重载的成员函数（以“=”,“+=”和“\*=”例，其他同理，其中“=”只能用运算符重载）：

myComplex& myComplex::operator = (myComplex& rhs){

real = rhs.real; imag = rhs.imag; return \*this;

}

myComplex& myComplex::operator +=(myComplex& rhs)

{ /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*this为对象指针，\*this则是指对象本身\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*所以可以调用重载后的复数加法运算\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\* -= 、\*= 、/= 、同理 但必须要先声明对应的加减乘除\*\*\*\*/

\*this = \*this + rhs; return \*this;

}

myComplex& myComplex::operator \*=(myComplex& rhs){

\*this=\*this \* rhs; return \*this;

}

......

在类外定义类中声明的友元重载的成员函数（以“\*”，“/”和“>>”为例）：

myComplex operator \* (myComplex& rhs1, myComplex& rhs2){

return myComplex(rhs1.real \* rhs2.real - rhs1.imag \* rhs2.imag,

rhs1.real \* rhs2.imag + rhs1.imag \* rhs2.real);

}

myComplex operator / (myComplex& c1, myComplex& c2){

return myComplex((c1.real \* c2.real + c1.imag \* c2.imag)

/ (c2.real \* c2.real + c2.imag \* c2.imag),

(c1.imag \* c2.real - c1.real \* c2.imag)

/ (c2.real \* c2.real + c2.imag \* c2.imag));

}

istream& operator >>(istream& is, myComplex &rhs){

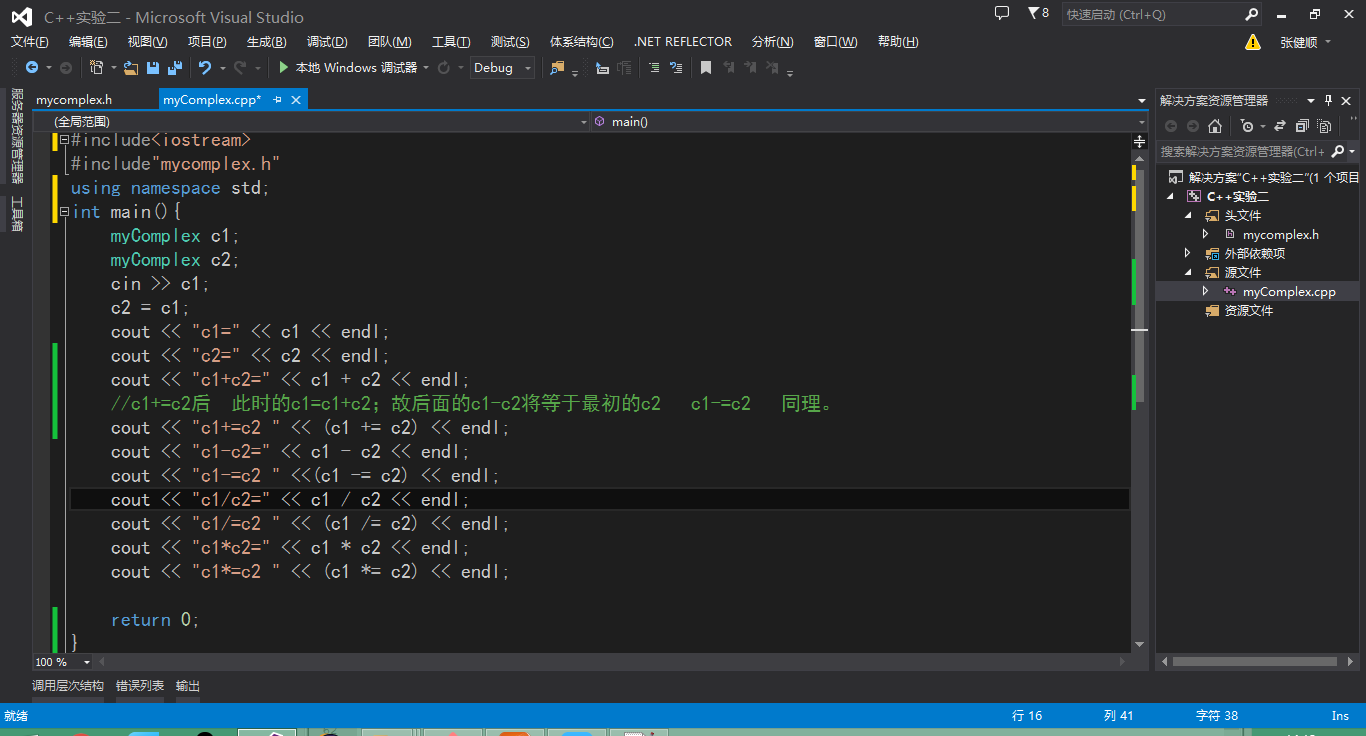
is >> rhs.real >> rhs.imag;

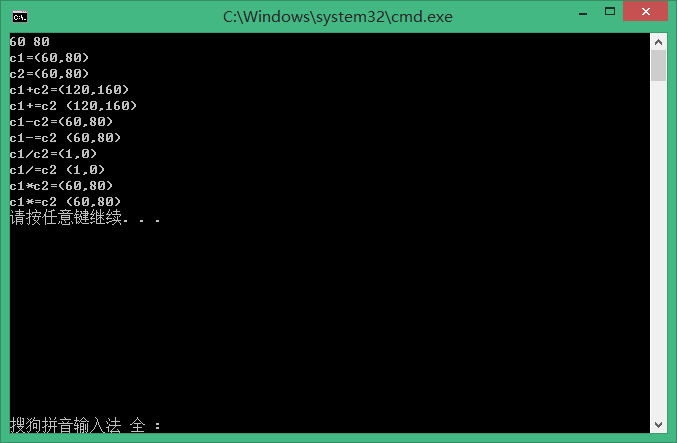
return is;

}

注意：“\*”和“/”的友元函数的重载返回值都是相应的构造函数初始化的临时对象，而输入输出的操作符重载后的返回值为相应的输入输出流。

（2）在myComplex.cpp编辑主程序，验证重载后的运算符号。调试运行程序,输入复数的实部和虚部后，显示结果.





c1的初始化是由用户自定义输入附属的实部和虚部部分，由于重载输入流的输入操作符，从而使得能够按照先实部后虚部的顺序来初始化c1，其中c2的初始化调用了复制构造函数，将c1复制给了c2。验证各类重载后的运算符当中，注意c1+=c2之后的运算里c1的值。

#include<iostream>

#include"mycomplex.h"

using namespace std;

int main(){

myComplex c1，c2;

cin >> c1;

c2 = c1;

cout << "c1=" << c1 << endl;

cout << "c2=" << c2 << endl;

cout << "c1+c2=" << c1 + c2 << endl;

//c1+=c2后此时的c1=c1+c2；故后面的c1-c2将等于最初的c2，c1-=c2同理。

cout << "c1+=c2 " << (c1 += c2) << endl;

cout << "c1-c2=" << c1 - c2 << endl;

cout << "c1-=c2 " <<(c1 -= c2) << endl;

cout << "c1/c2=" << c1 / c2 << endl;

cout << "c1/=c2 " << (c1 /= c2) << endl;

cout << "c1\*c2=" << c1 \* c2 << endl;

cout << "c1\*=c2 " << (c1 \*= c2) << endl;

return 0;

}

1. **总结及心得体会：**

（1）运算符重载的成员函数以及友元重载的成员函数的区别有：

A.运算符重载的成员函数参数个数为原操作数个数减一，而友元重载的成员函数的参数个数为原操作数个数；

B.成员函数有this指针，友元函数无this指针；

C.成员函数运算的结果可以作为左值使用，不是一个新产生的结果，而友元重载的运算结果是一个新产生的值，并且这个值不能作为左值使用。

1. 对于运算符“=，（），[]，->”只能作为成员函数；
2. 重载函数：通过函数的参数表唯一标识并且来区分函数的一种特殊的函数。
3. **对本实验过程及方法、手段的改进建议：**

“+=、-=、\*=、/=”运算符可以对之前“+、-、\*、/”的代码进行重用，从而达到简化代码的目的。重用的时候利用this指针，而\*this则是指对象本身。

**报告评分：**

**指导教师签字：**

**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。

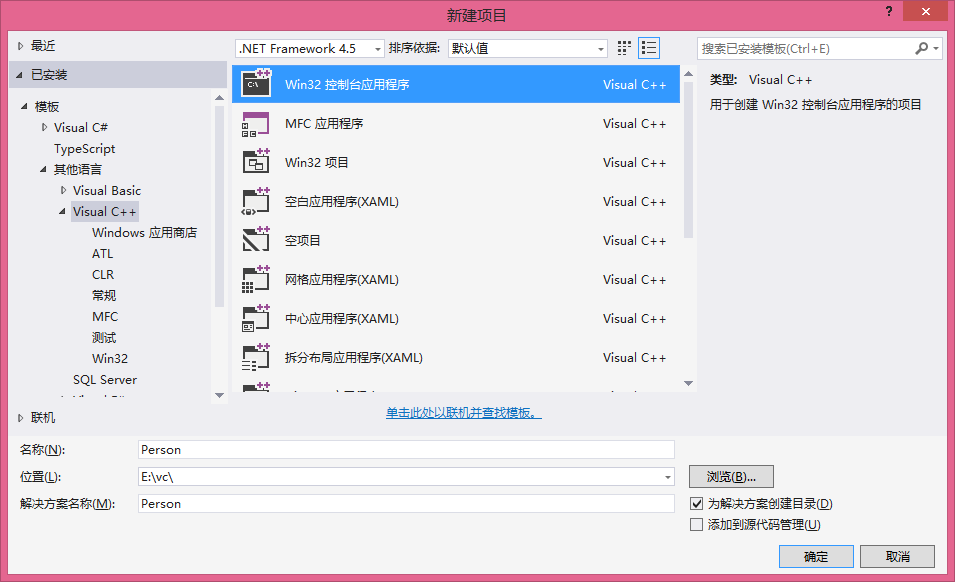
（6）掌握基类同名成员的访问方法。

1. **实验器材（设备、元器件）：**

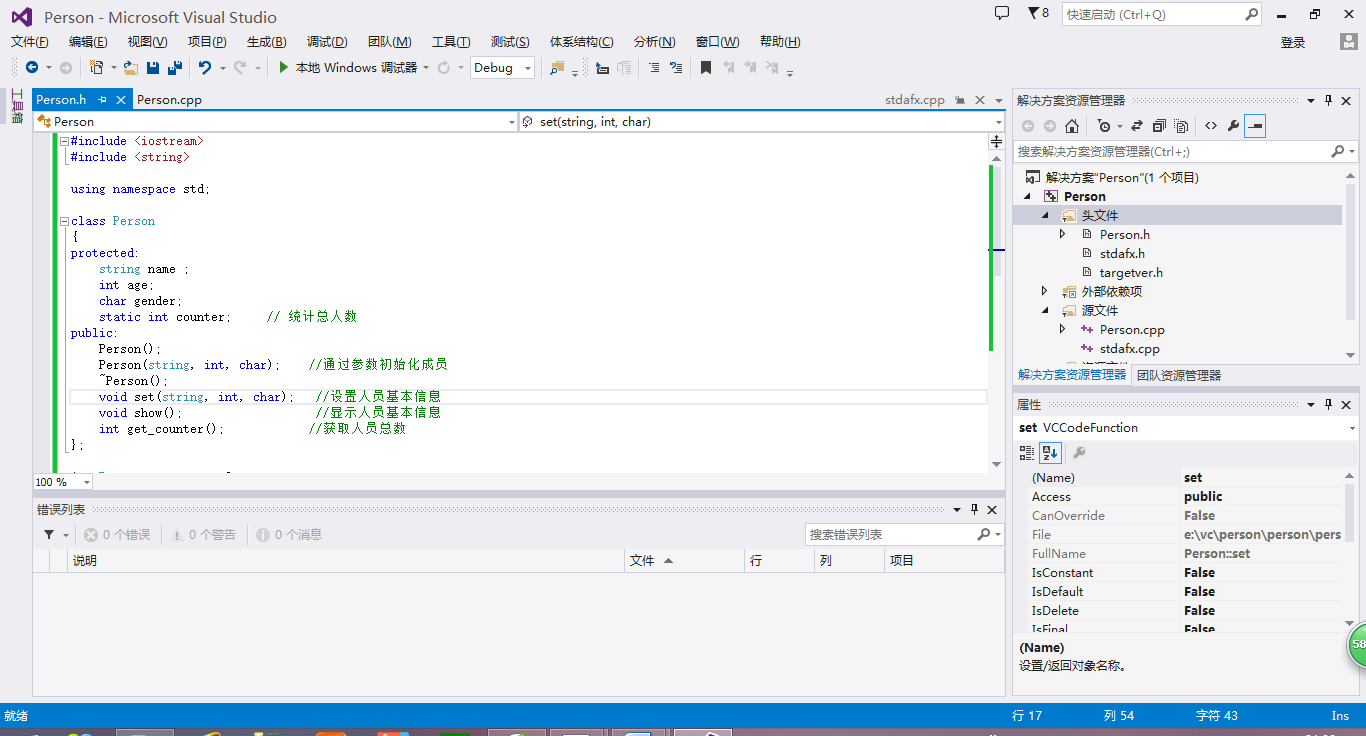
PC机一台，Windows 8.1，Visual Studio 2013

**六、 实验步骤：**

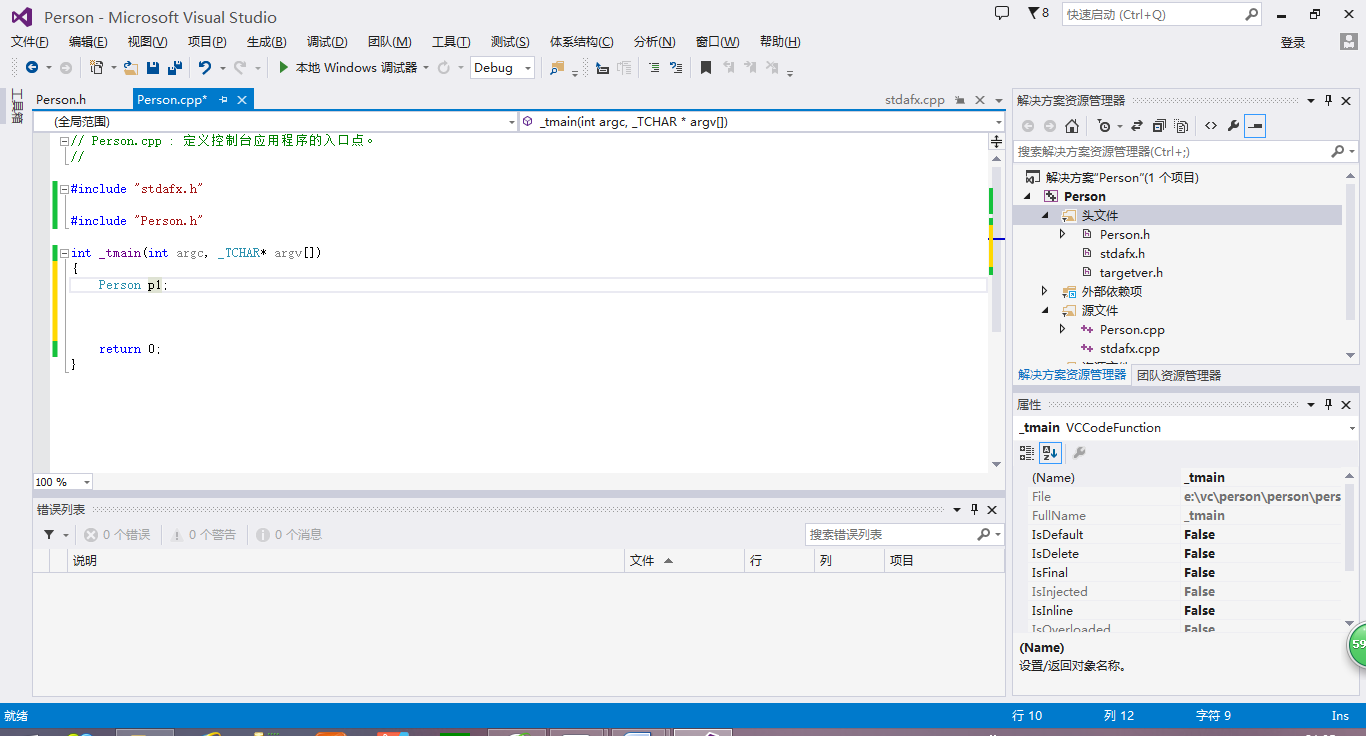
(1) 建立工程，项目名称设置为“Person”，类型选择“Win32控制台应用”。



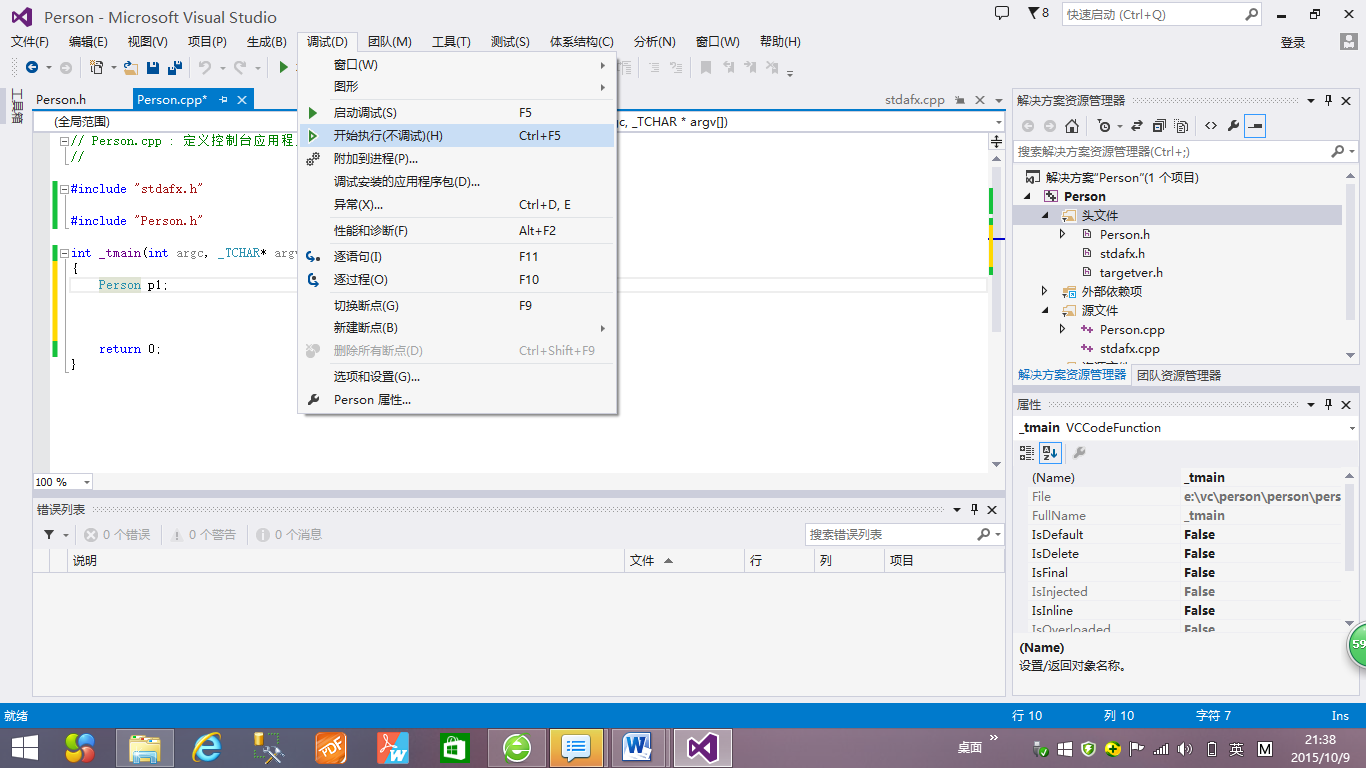
1. 按应用程序向导选择下列各项：控制台应用程序(Console application)，预编译头(Precompiled header)，安全开发声明周期(SDL)检查；
2. 添加头文件Person.h
3. 在头文件Person.h中定义Person类及其他各派生类，并实现各类的成员函数。



（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。

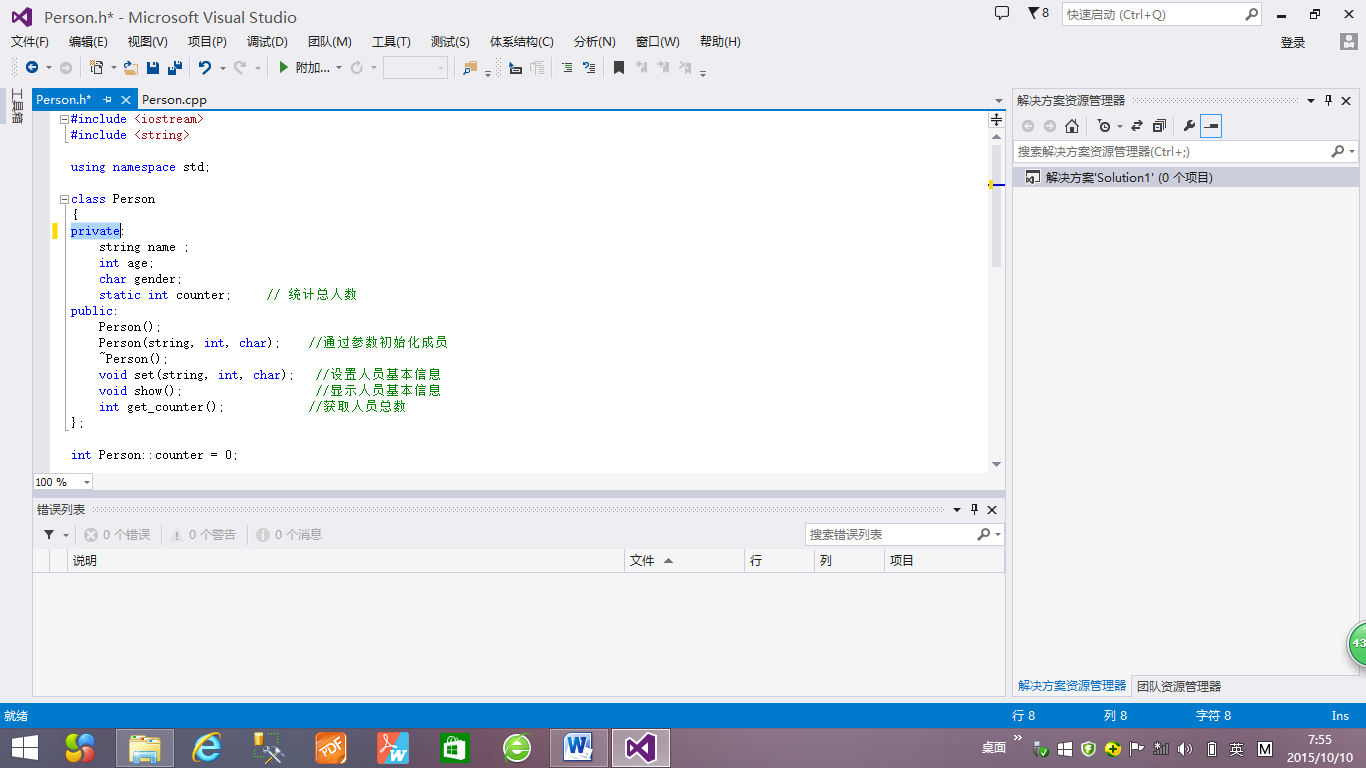


（6）选择菜单“调试”->开始执行(不调试)，运行程序，若有错误，需按错误提示逐个修改。

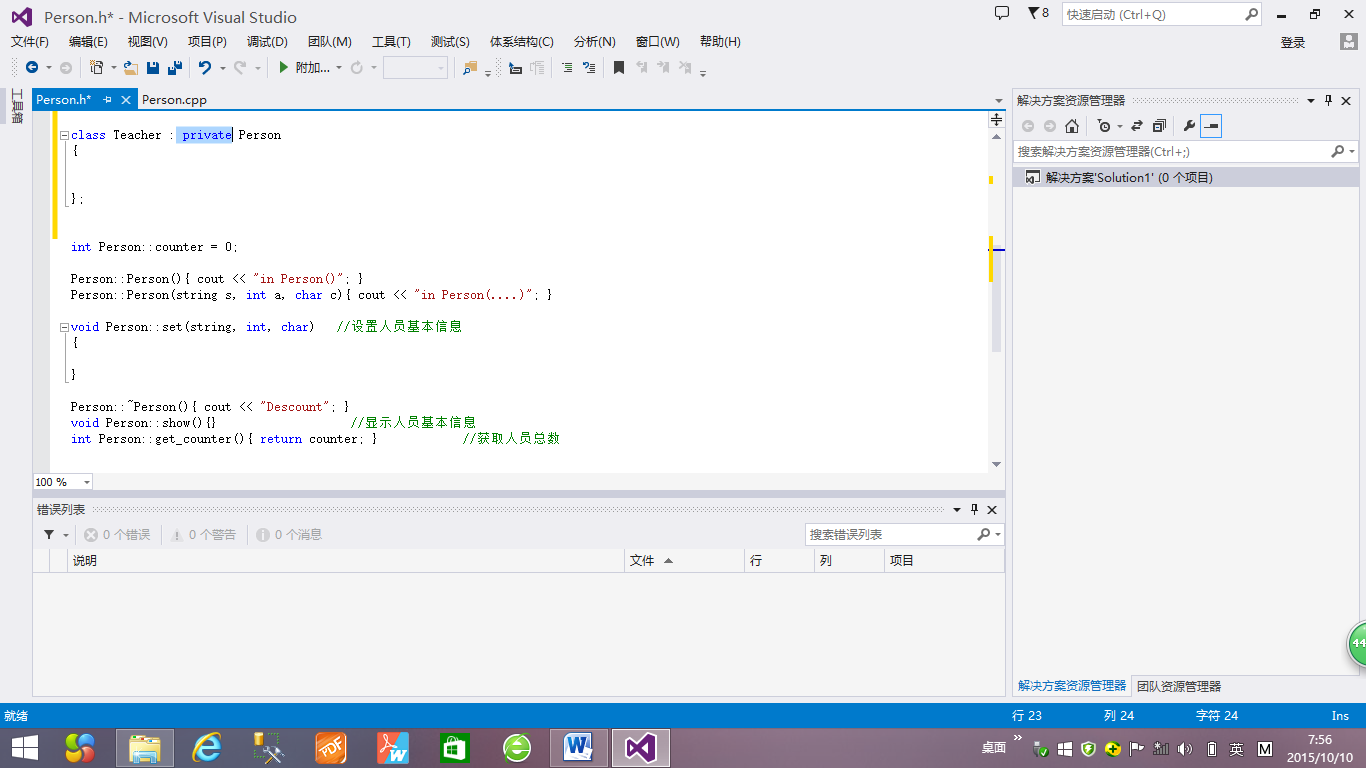


（7）运行程序，验证各类的功能。

（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。

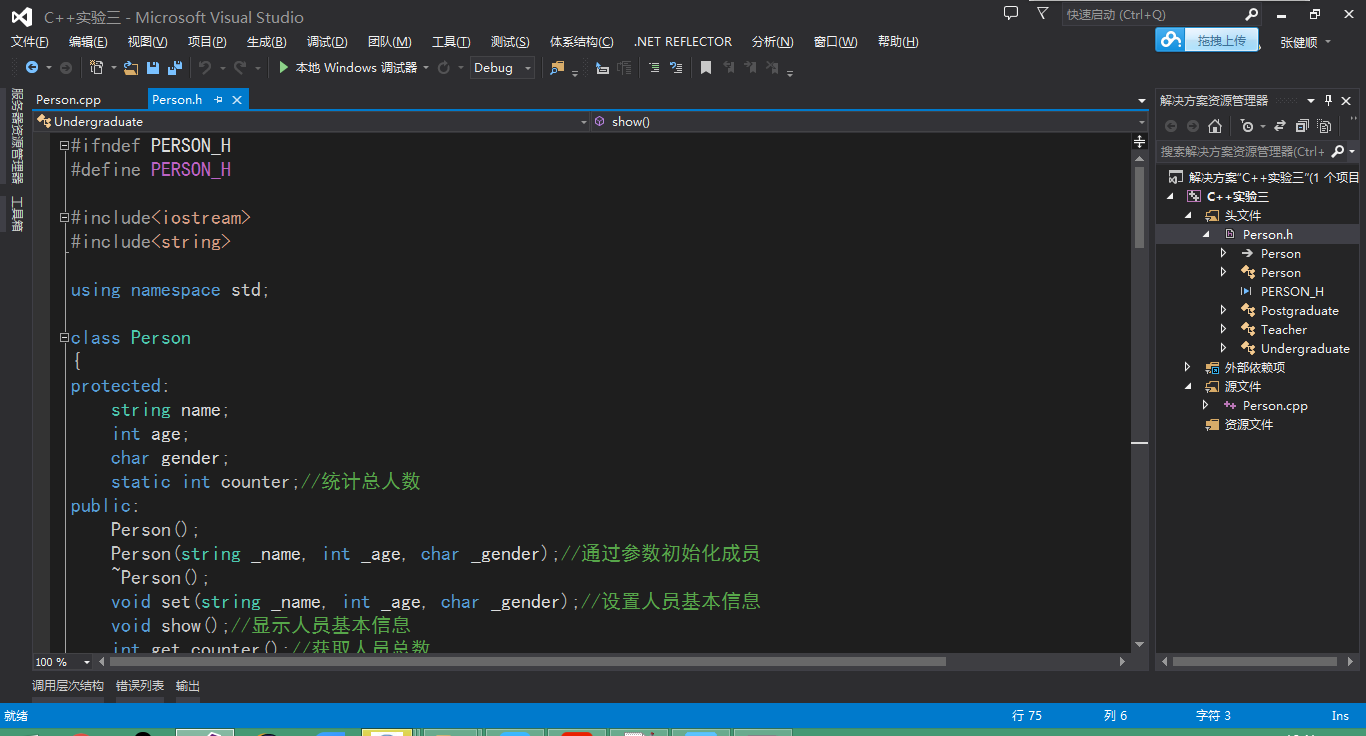


（9）将继承方式分别修改为private继承和Protected继承，运行程序，分析不同继承方式对基类成员在派生类中访问属性的影响。



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){

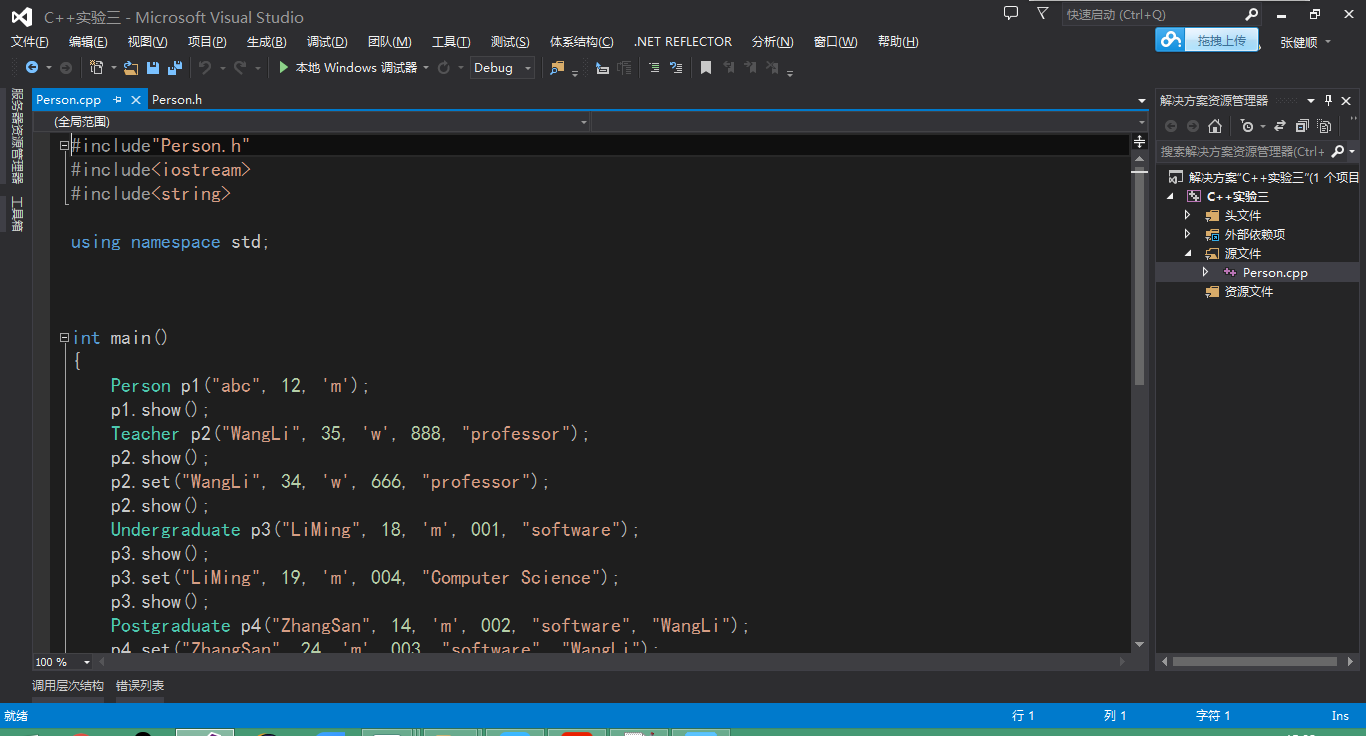
Undergraduate::set(\_name, \_age, \_gender, \_ID, \_depart);

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");

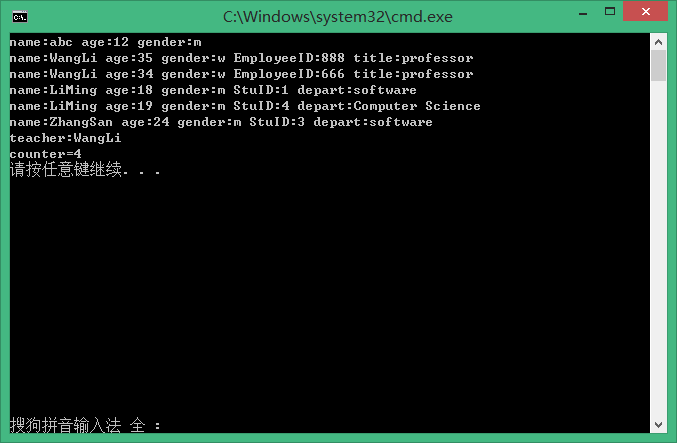
p4.set("ZhangSan", 24, 'm', 003, "software", "WangLi");

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

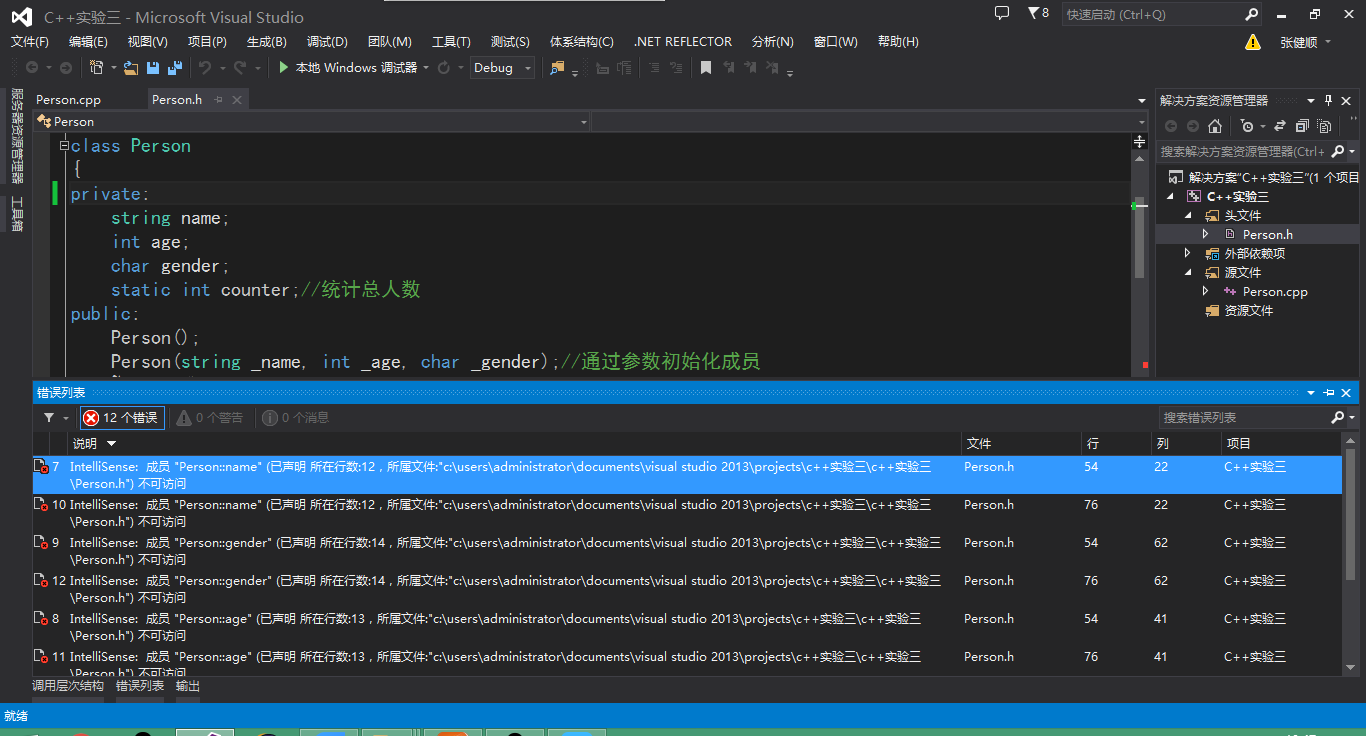
private:

string name;

int age;

char gender;

static int counter;//统计总人数



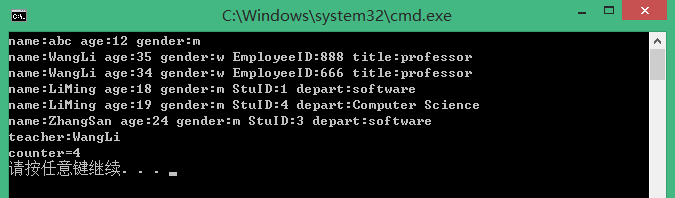
属性声明修改为private之后报错

（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。

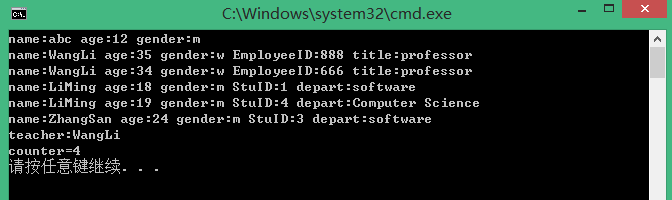
将继承方式分别修改为private继承和Protected继承，运行程序，分析不同继承方式对基类成员在派生类中访问属性的影响。

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：**

**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：16.11.15**

1. **实验名称：多态与虚函数程序设计**
2. **实验学时：4**
3. **实验目的：**

通过实验练习，让学生理解静态联编和动态联编两种不同的多态机制；理解赋值兼容原则及其实现方法；理解虚函数与普通函数重载的区别; 掌握通过虚函数实现运行时多态的方法;掌握纯虚函数的概念，以及抽象类的使用方法。

1. **实验内容：**

编写小型公司的工资管理程序。公司主要有3类人员：经理、技术人员和销售人员。要求存储并显示各员工的编号、姓名和月薪等基本信息。其中各类员工的月薪计算方法如下：

经理：固定月薪8000元；

技术人员：50元/小时；

销售人员：当月销售额的3%

编程完成如下任务：

（1）根据如上要求，设计基类Employee，包含各类员工共有的基本信息，以及静态数据成员totalNo，用于自动计算员工编号（员工编号从1开始）；基类定义构造函数和析构函数，另外还需要定义计算员工月薪和显示员工信息的成员函数，以规范各派生类的行为。 Employee类声明如下：

（2）分析其余各类人员的特征，设计合理的继承结构，并在Employee类的基础上派生出其余各类人员，每个类需要对从基类继承的虚函数进行重新定义。

class Employee

{

protected:

int No;

string name;

float salary;

static int totalNo； // 自动计算员工编号

public:

Employee (); //自动计算员工编号，姓名从键盘输入，工资初值为0

~Employee ();

virtual void pay ( )=0; //计算月薪

virtual void display( ) = 0; //显示人员信息

}；

（3）在主函数中通过基类指针指向各派生类对象，通过基类指针访问各派生类对象的成员函数，通过程序运行结果观察虚函数如何实现运行时多态。

（4）将基类的虚函数修改为普通成员函数，通过运行结果分析虚函数与普通函数重载的区别。

要求：

（1）理解静态联编和动态联编；

（2）理解赋值兼容原则；

（3）理解虚函数与普通函数重载的区别;

（4）掌握虚函数实现运行时多态的方法;

（5）掌握纯虚函数的概念，以及抽象类的使用方法。

实验思考

（1）虚函数和普通函数重载的区别；

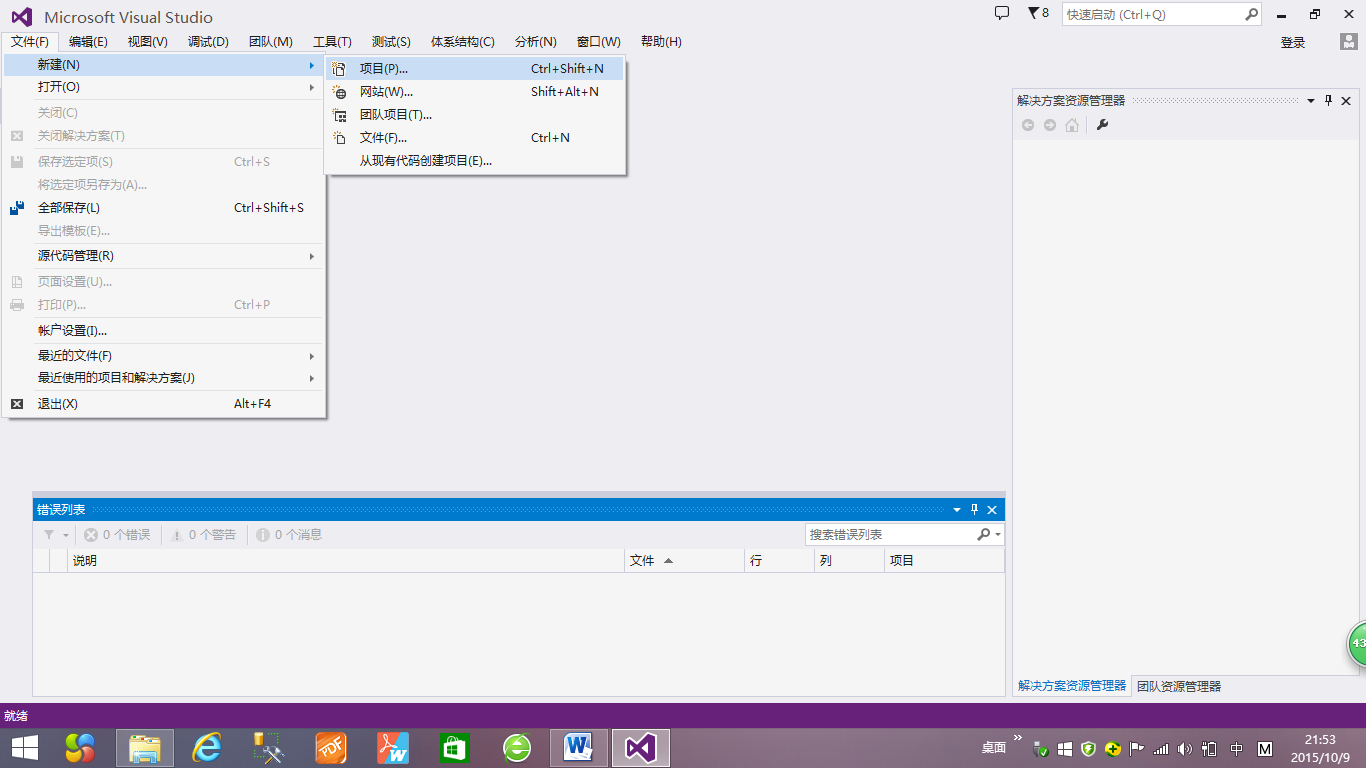
（2）虚函数如何实现多态。

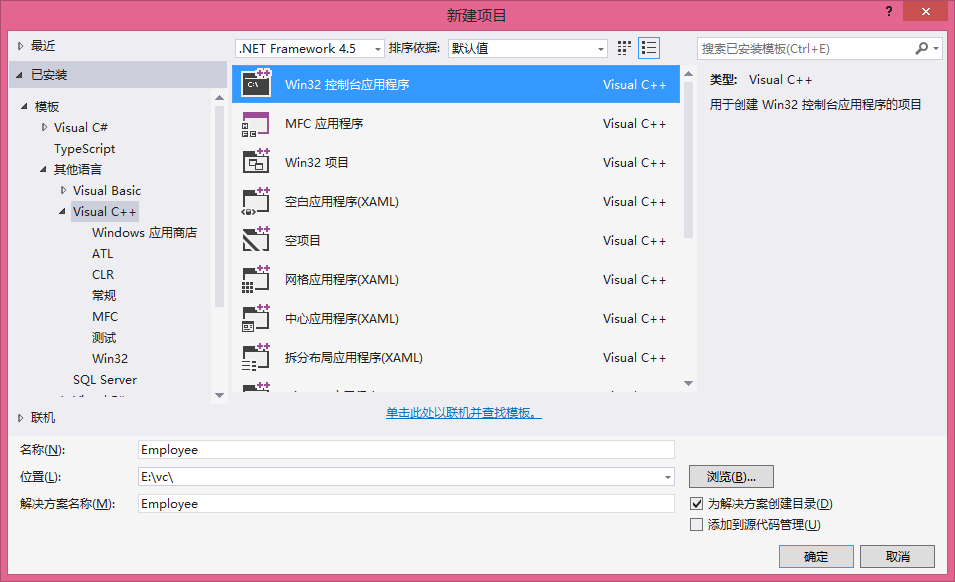
1. **实验器材（设备、元器件）：**

PC机一台，Windows 8.1，Visual Studio 2013

1. **实验步骤：**

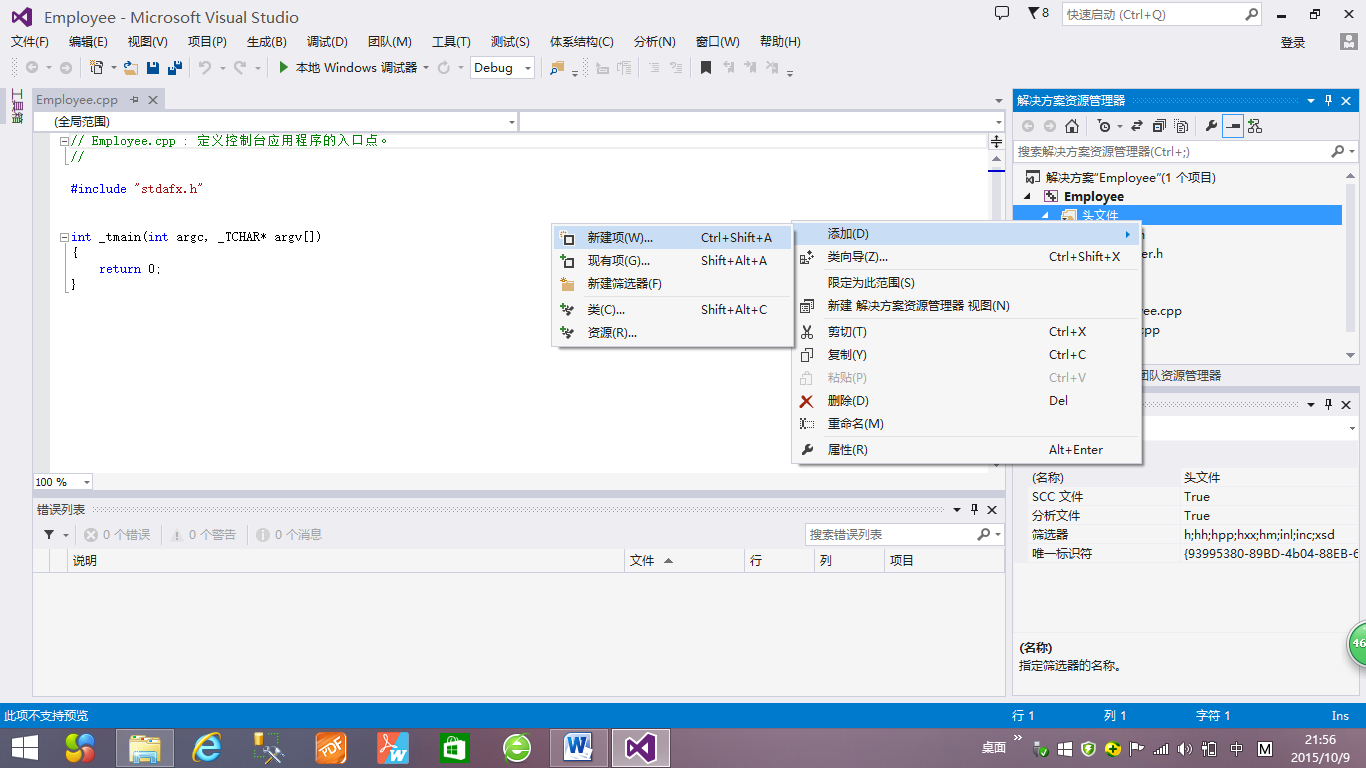
(1) 新建工程，名称设置为“Employee”，类型选择“Win32控制台应用”。

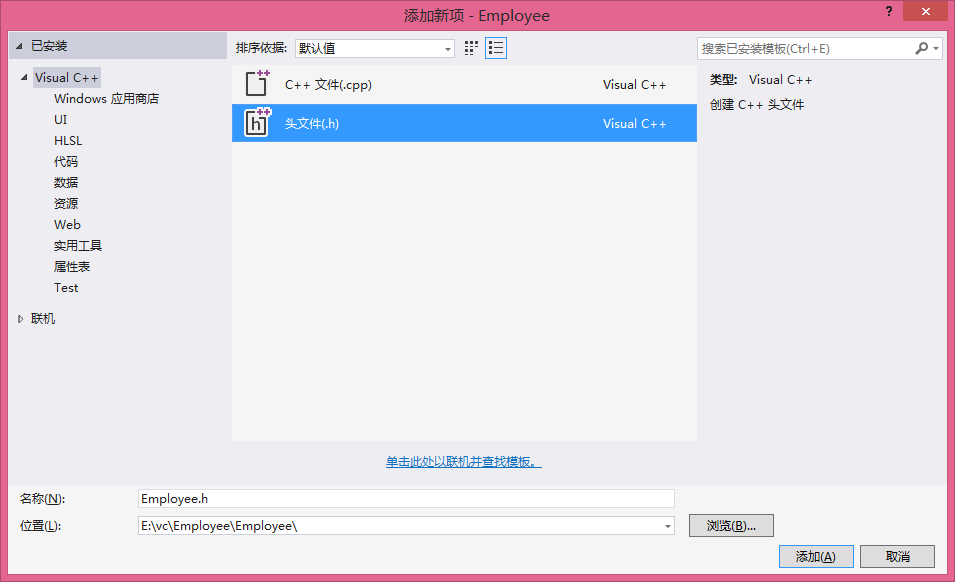




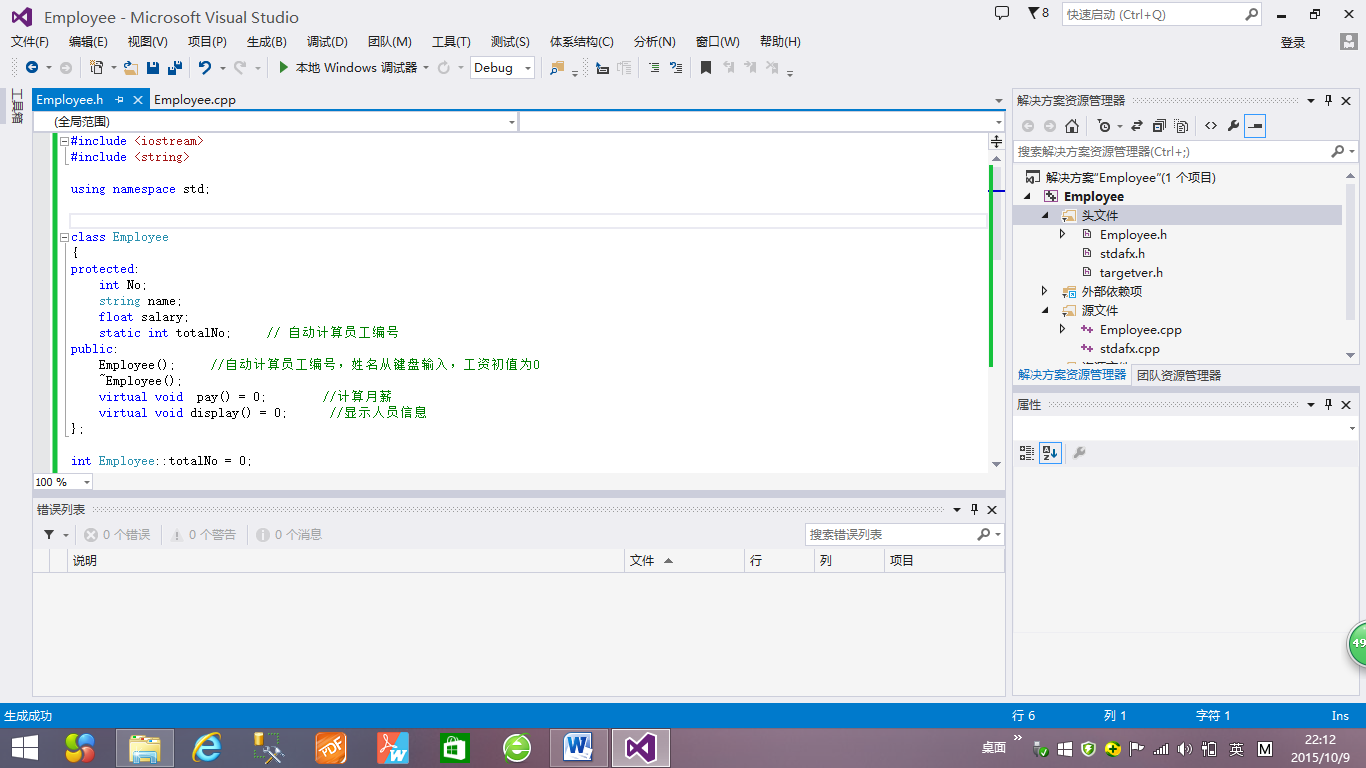
(2) 按应用程序向导选择控制台应用程序(Console application)，预编译头(Precompiled header)，安全开发声明周期(SDL)检查。

（3）参照下列图例，添加头文件Employee.h

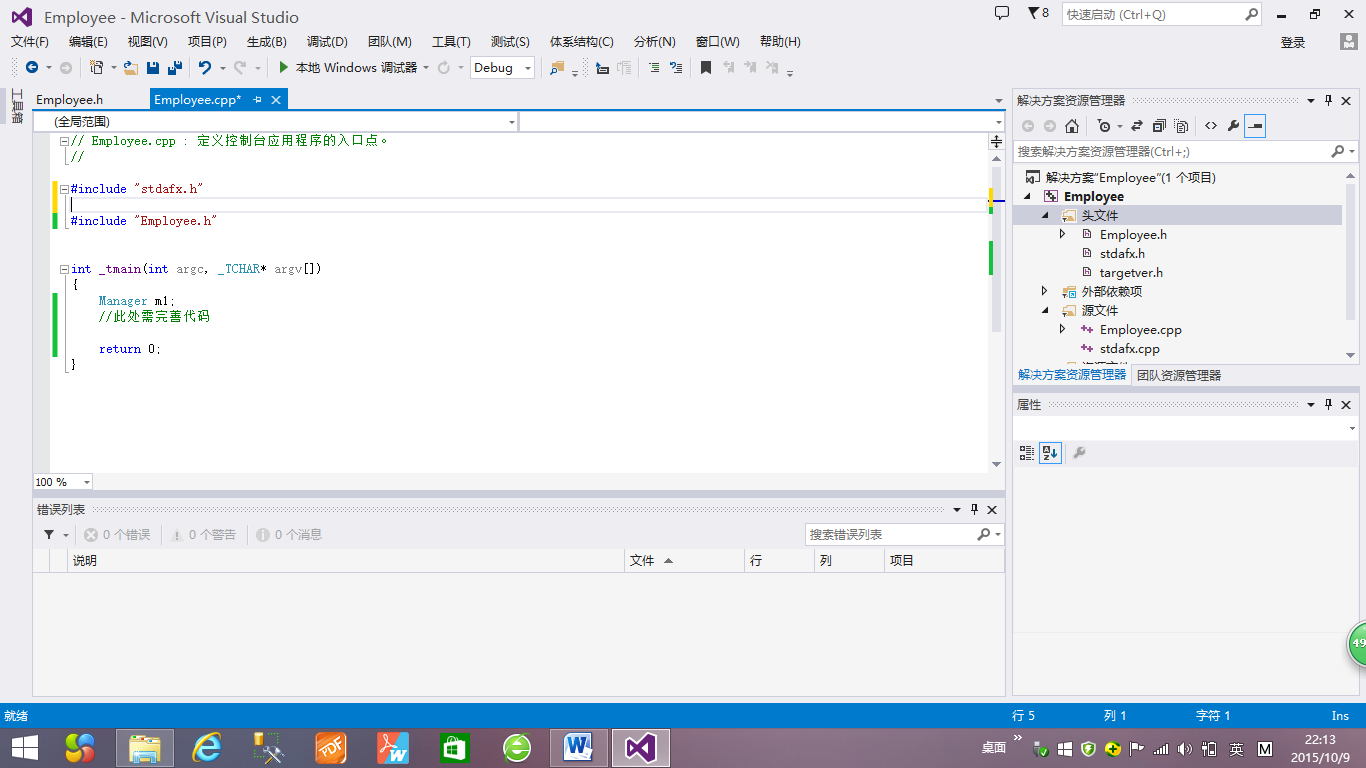




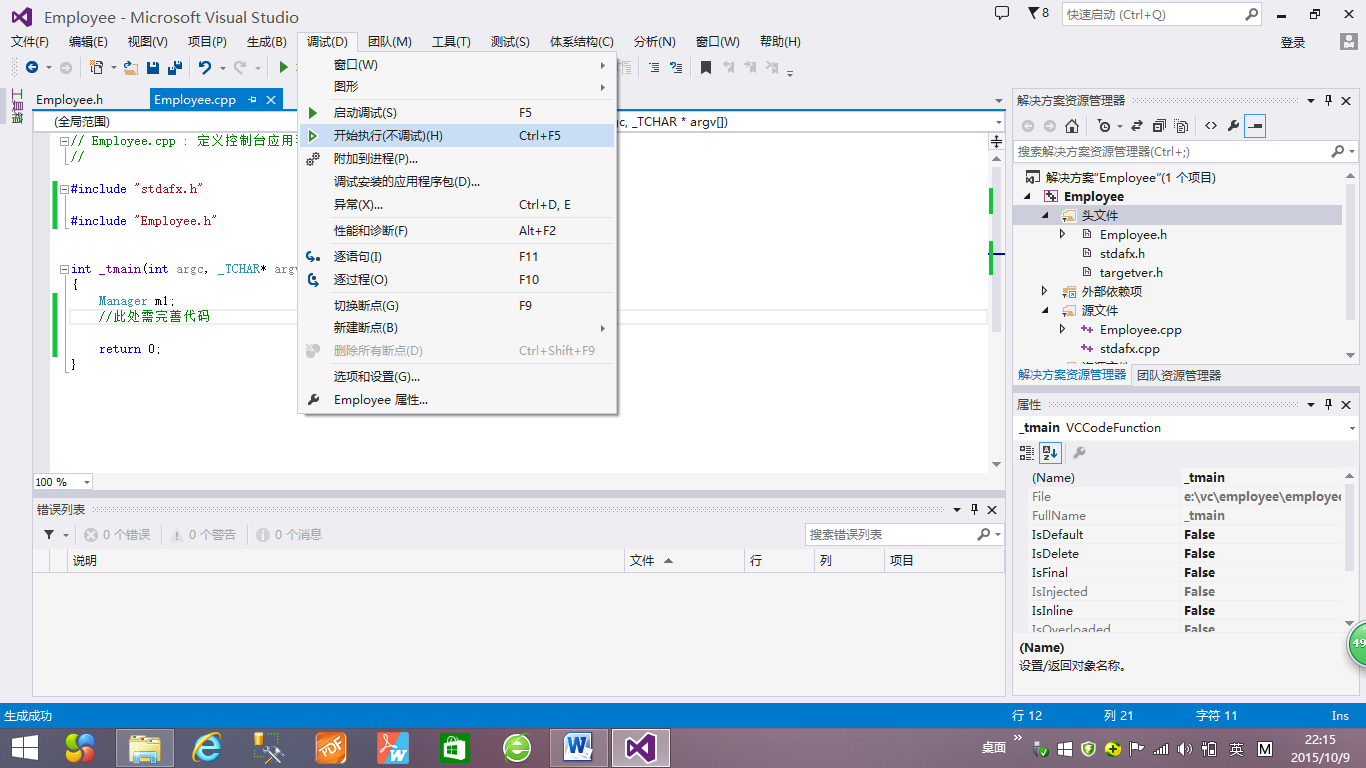
（4）在头文件中定义Employee类，并在此基础上派生出其他各类人员的类型，实现各类的成员函数。



（5）在Employee.cpp中编辑主程序。

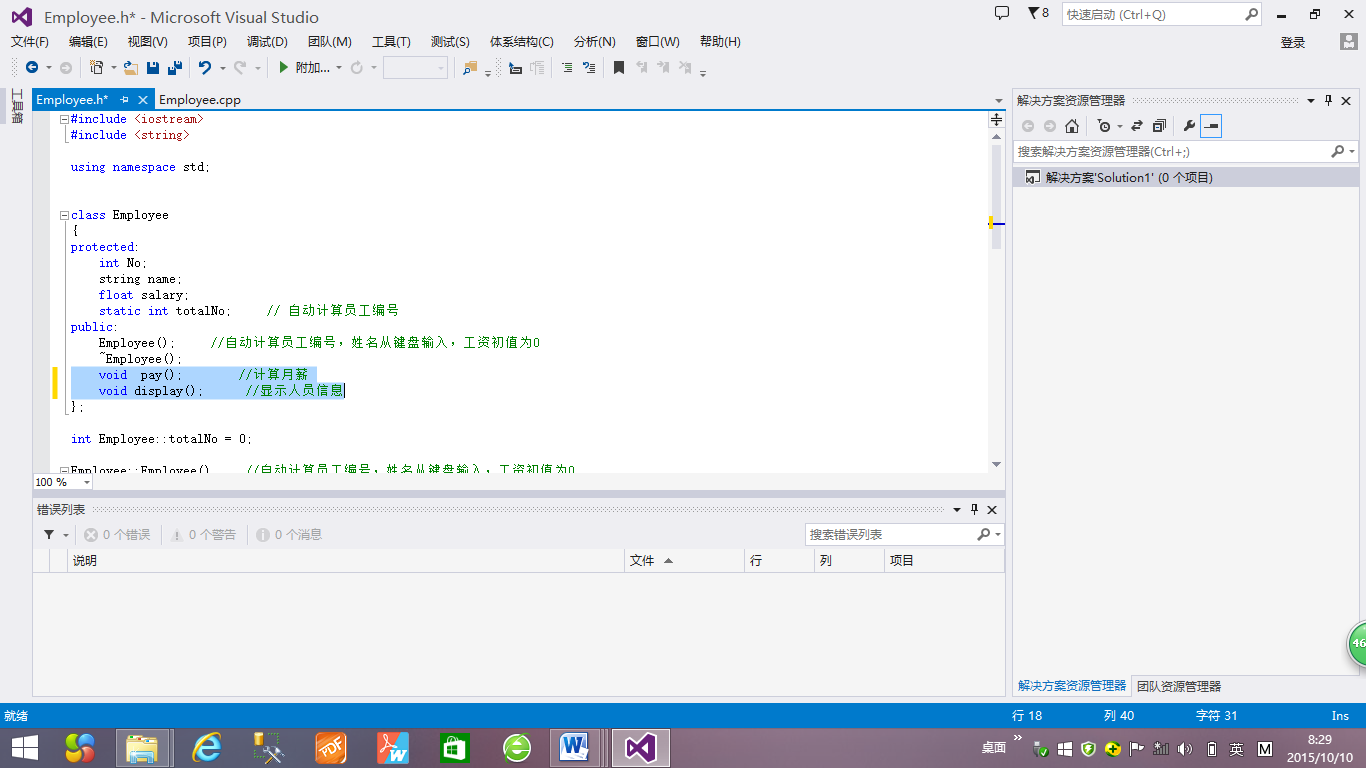


（6）调试程序，修改编译/连接错误



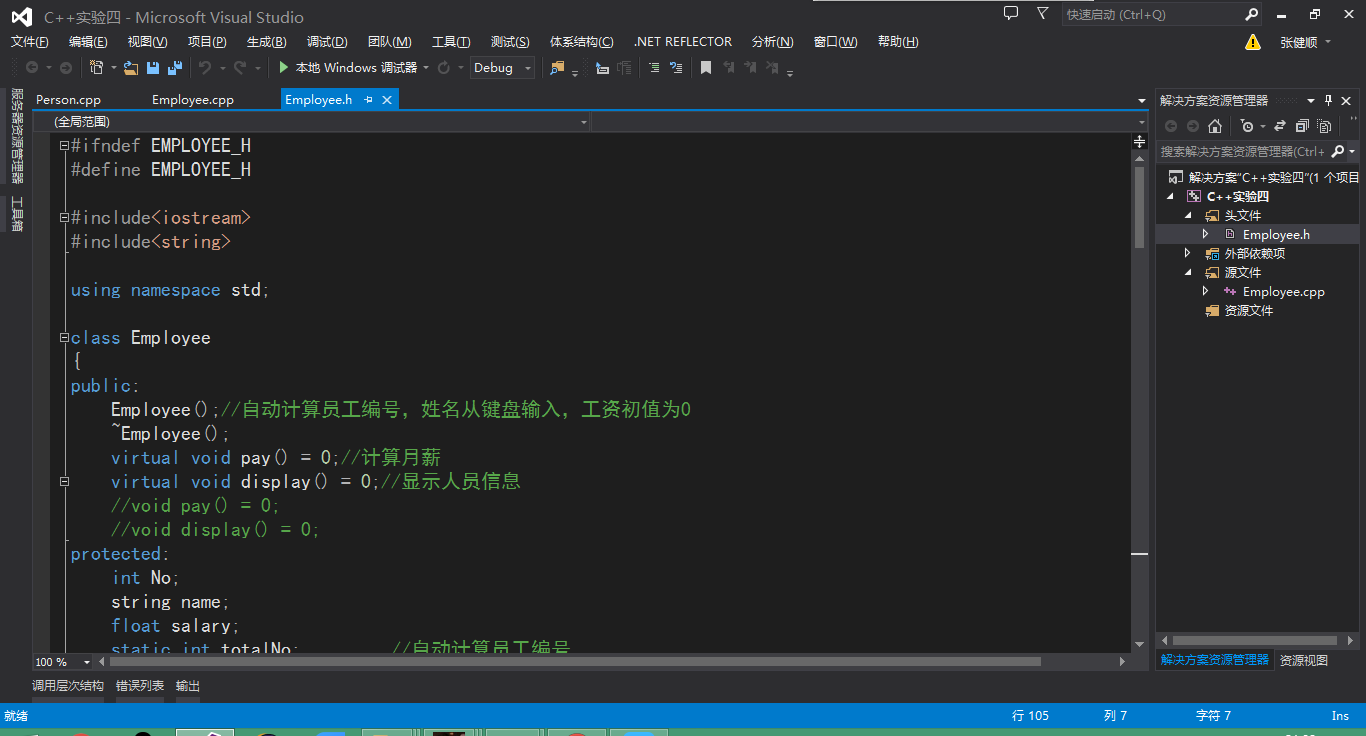
（7）运行程序，验证虚函数的多态性。

（8）将基类虚函数修改为普通成员函数，运行程序，分析虚函数与普通函数重载的区别。



**七、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）创建工程，创建头文件；



1. 在头文件中定义Employee类，并在此基础上派生出其他各类人员的类型，实现各类的成员函数。其中将计算月薪和输出人员信息的函数定义为虚函数。

class Employee

{

public:

Employee();//自动计算员工编号，姓名从键盘输入，工资初值为0

~Employee();

virtual void pay() = 0;//计算月薪

virtual void display() = 0;//显示人员信息

protected:

int No; string name; float salary;

static int totalNo; //自动计算员工编号

};

int Employee::totalNo = 0;

Employee::Employee(){

salary = 0; cout << "Please enter the name:";

cin >> name; totalNo++; No = totalNo;

}

Employee::~Employee(){}

void Employee::pay(){

cout << "基类pay函数执行" << endl;

}

void Employee::display(){

cout << "基类display函数执行" << endl;

}

其中构造函数可以自定义输入人员姓名，同时定义了静态变量员工编号，将会在每一次Employee构造函数的调用中进行编号的自动计算。

派生类：

A、经理（Manager）类（定义私有成员变量monthSalary）

class Manager : public Employee

{

public:

Manager() :Employee(){

monthPay = 8000; //经理固定月薪8000

}

void pay(){

salary = monthPay;//月薪

}

void display(){

cout << "编号为" << No << "的经理人员 " << name

<< " 本月薪水为" << salary << endl;

}

private:

float monthPay;

};

B、技术人员（Technicist）类(定义了私有成员变量hours和hourPay)

class Technicist : public Employee

{

public:

Technicist() :Employee(){

hourPay = 50;//技术人员：50元每小时

}

void pay(){

cout << "Please enter the work hours of technicist:";

cin >> hours;

salary = hours\*hourPay;//工作时长\*每小时的工资

}

void display(){

cout << "编号为" << No << "的技术人员 " << name << " 本月薪水为" << salary << endl;

}

private:

float hours; //工作时长

float hourPay; //每个小时支付薪水

};

对于技术人员，除开自定义输入的个人信息以外还将自定义输入工作时长。

C、销售人员（SalesMan）类（定义了私有成员变量TotalSale和percantage）

class SalesMan : public Employee

{

public:

SalesMan() :Employee(){

percantage = 0.03;//销售人员：当月销售额的百分之三

}

void pay(){

cout << "Please enter the total sales of the salesman:";

cin >> TotalSale;

salary = TotalSale\*percantage;//销售额\*提成比例

}

void display(){

cout << "编号为" << No << "的销售人员 " << name << " 本月薪水为" << salary << endl;

}

private:

float percantage; //提成比例

int TotalSale; //销售额

};

1. 在Employee.cpp中编辑主程序

int main()

{

cout << "Manager:";

Manager manager;

cout << "Technicist:";

Technicist technicist;

cout << "SalesMan:";

SalesMan salesman;

Employee \*p1 = &manager,\*p2=&technicist,\*p3=&salesman;

p1->pay();

p1->display();

p2->pay();

p2->display();

p3->pay();

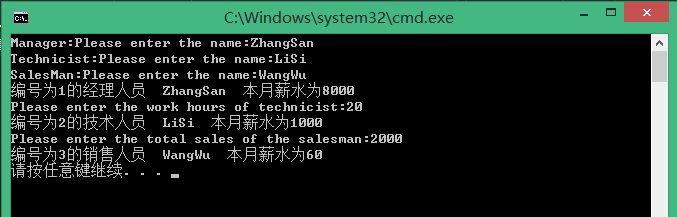
p3->display();

return 0;

}

p1,p2,p3均为父类指针，然后分别指向三个不同派生类的对象

**运行结果：**



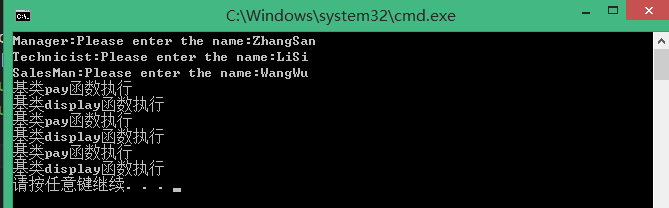
由于调用到了派生类各自的构造函数，所以需要自定义输入Manager、Technicist、SalesMan的姓名，又因为将基类的pay()和display()函数声明为了虚函数，所以再利用基类指针声明派生类对象并调用相应函数的的时候会调用派生类中自定义的函数，所以需要再自定义输入技术人员工作时长以及销售人员销售额。按照程序中设定好的值进行相关计算。

（4）将基类虚函数修改为普通成员函数，运行程序，分析虚函数与普通函数重载的区别。

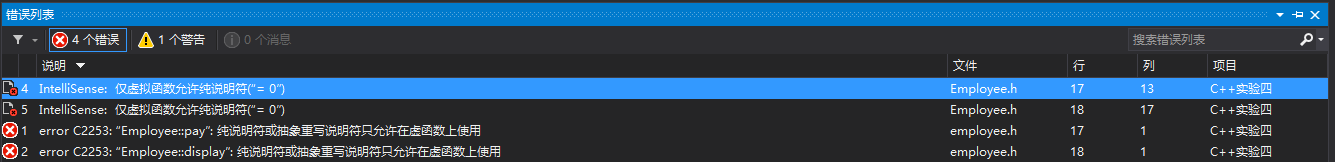
void pay() ; //注意此时不能再将纯说明符加入到函数末尾

void display() ; //仅虚拟函数允许使用纯说明符

**运行结果：**



由运行结果可知：

由于声明为父类指针的作用，而函数为普通成员函数，故调用对象相应的构造函数的时候会调用父类的构造函数和析构函数，由于构造函数和虚构函数是在派生类的构造函数中和虚构函数中调用过，但无法调用派生类各自相应的pay()函数和display()函数，只能调用基类的pay()和display()函数。

**八、总结及心得体会：**

（1）虚函数与普通函数重载的区别：

A.函数重载可以用于非成员函数和类的成员函数，而虚函数只能用于类的成员函数

B.函数重载可用于构造函数，而虚函数不能用于构造函数

C.重载的函数必须具有相同的函数名，函数类型可以相同也可以不同，但函数的参数个数和参数类型二者中至少有一个不同，否则在编译时无法区分。而虚函数则要求同一类族中的所有虚函数的函数名，函数类型，函数的参数个数和参数类型都全部相同，否则就不是重定义了，也就不是虚函数了

D.函数重载是在程序编译阶段确定操作的对象的，属于静态关联。虚函数是在程序运行阶段确定操作对象的，属于动态关联。

（2）纯虚函数与抽象类

纯虚函数是一个在基类中说明的虚函数，它在该基类中没有定义，要求任何派生类都必须定义自己的版本。

如果一个类至少有一个纯虚函数，那么就该称该类为抽象类。

**九、对本实验过程及方法、手段的改进建议：**

可以通过在Employee类中设置纯虚函数的方法将Employee类定义为抽象类，当然也要对其中的函数进行相应的改造。

**报告评分：**

**指导教师签字：**

**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：16.10.25**

1. **实验名称：基于MFC的C++桌面应用开发**
2. **实验学时：4**
3. **实验目的：**

充分运用所学的C++的数据封装、继承与派生、多态等全部核心内容，补充完整一个基于MFC的应用，使学生能够基本掌握OOA、OOD方法；熟练掌握OOP方法；初步了解基于Windows平台的桌面图形化应用的开发过程，掌握编程工具的使用；初步掌握建模工具的使用；基本掌握阅读、调试程序的能力。

**四、实验内容：**

一位小学教师Ken希望完成这样的任务：针对于小学生正在学习四边形(quadrangle)的特性，编写一个小软件，能够随机在屏幕上显示矩形(rectangle)、正方形(square)、平行四边形(parallelogram)、梯形(trapezoid)和菱形(diamond)五种形体之一，同时显示该形体的特性和关键数据（随机产生），学生复习形体的特性，然后根据给出的关键数据计算形体的面积，软件判断其结果的正确性。在学习过程中，软件记录产生的每一个形体，在学生选择不再继续后，将其学习的过程重放一遍，用以重温，加深印象。

根据上述描述，需要完成：

1. **必做内容**

根据Ken老师的要求，需要至少编写六个类：

* Rect //注意：类名不要使用Rectangle
* Square
* Parallelogram
* Trapezoid
* Diamond
* List

（注：这六个类必须以上述名字命名）

其中，前五个类用于描述五种形体。五种形体不用顶点坐标的表示形式，而只是简单地用它们的特征值表示。例如：矩形、平行四边形用长和高表示；正方形用边长表示；梯形用两条平行边长和高表示；菱形用两条对象线长表示。要求为这五种形体编写相应的类，每个类的设计要求如下：

1. Quadrangle类必须成为抽象类，是其它形体类的祖先。它拥有如下成员：

|  |  |  |
| --- | --- | --- |
| **成员** | **类型** | **是否纯虚** |
| **name** | 数据 | / |
| **area** | 函数 | 是 |
| **draw** | 函数 | 是 |
| **what** | 函数 | 可选 |

1. 其它形体类之间的继承关系请自行拟定。其中，Parallelogram类必须拥有如下虚成员：

* Width(); //返回宽
* Height(); //返回高

1. 每个形体类必须完成如下操作：
2. 设置标志名属性name。五种形体的标志名必须是Parallelogram、Rectangle、Diamond、Trapezoid和Square之一（第一个字母大写，其余小写）；
3. 重载area()成员；
4. 重载draw()成员；
5. 重载Width()成员以返回宽；
6. 重载Height()成员以返回高；
7. 提供Width2()成员以返回第二条平行边长（仅对梯形）
8. List类必须拥有如下成员函数：

* size()；//返回列表中的节点数
* push\_back()；//将数据添加到列表末尾
* operator[]；//返回指定下标的数据
* traverse()；//遍历，需要一个访问函数作为参数
* pop\_back() //用来删除列表的最后一个节点，其原型为：

void pop\_back();

1. 重温过程实际上就是遍历List类对象的过程。
2. **选作内容**

List类实际上是一种称为“容器”的类。除了容纳形体指针，其实它还可以容纳任何类型的对象。在本选作内容中，

1. 请将List类该为模板类，使其能容纳任何类型的对象；
2. 请为List类编写迭代器。要求如下：

* 该迭代器的类名为Iterator;
* 该迭代器必须重载如下运算符：
* ++：迭代器后移
* !=：两个迭代器比较
* \*：返回迭代器指向节点的数据域中的值，即quad指针。
* =：两个迭代器的复制
* +。该运算符函数的原型为：

Iterator operator+(int i);

其功能为：假设当前迭代器（的内部指针）指向了第k个节点（从0开始计数），那么+运算符返回一个迭代器，该迭代器是在当前迭代器的基础上向后移动i个位置得到的，即该迭代器指向了第k+i个节点。例如：

Iterator itr = list.begin() + 5;

那么迭代器itr就指向了从列表的头节点往后数第5个结点。

为List类编写begin()和end()成员。

要求：

1. 熟练掌握面向对象的核心知识：封装、继承、多态，基本掌握泛型编程；
2. 基本掌握OOA、OOD方法；熟练掌握OOP方法；
3. 初步了解基于Windows平台的桌面图形化应用的开发过程，掌握编程工具的使用；

4）初步掌握建模工具的使用。

1. **实验器材（设备、元器件）：**

PC机一台，Windows 8.1，Visual Studio 2013

1. **实验步骤：**
2. 先在纸上完成系统设计、类设计；
3. 使用任意一款建模工具建立系统的模型，至少画出类图

根据设计，编写各种形体类代码。建议：

1. 请将所有类型声明放在同一个.h文件中，所有类的实现放在同一个.cpp文件中；
2. 先不将以上代码整合到系统中，单独编写测试模块完成所有类的功能测试。
3. 五种形体的名字标识要求使用如下字符串（单词的第一个字母大写）：
   * 四边形：Quadrangle
   * 平行四边形：Parallelogram
   * 梯形：Trapezoid
   * 矩形：Rectangle
   * 菱形：Diamond
   * 正方形：Square

注：以上只是用于属性name的名字表示，而非类名！

1. 在用于测试的main函数中：
   * 不能预先定义形体对象，而采用随机的方式生成形体。这需要使用两个函数：

* 随机数发生器初始化：srand(unsigned(time(NULL))); 该函数只需调用一次；这次调用最好放在main()函数中；
* 随机数发生器：rand();该函数产生0-INT\_MAX之间的伪随机正整数。请思考：如何产生合理范围内的整数？

为了使用以上函数，需要包含以下几个头文件：

cstdlib

ctime

* + 形体的所有关键数据（例如宽和高）也采用随机的方式生成

1. 程序流程
   * 程序首先只考虑学习模式。在该模式下：
     1. 随机产生一个形体及其关键数据，然后“画”出这个形体（即简单地显示这个形体的关键数据）。此后要求学习者输入面积，程序判断输入是否正确，如果不正确则要求继续输入，直到正确为止；
     2. 此后程序询问学习者是否继续。如果是，则重复上一步；
     3. 否则，程序进入复习模式。
   * 暂时不考虑复习流程。

继续第二次实验的编码过程，直到完成所有编码和测试工作。

仿照第二次实验的流程进行List类的编码和测试。

1. 对于List类重载[]说明

如果有：List list;

那么list[2]返回list对象内部链表中第2个（从0开始计数）节点中quad域存储的数据，亦即类型为QUADPTR的指针。

提示：需要用到与遍历相似的算法。

建议：该函数的原型为：

QUADPTR& operator[](int index);

1. 程序流程的完善
   * 程序首先进入学习模式。在该模式下：
2. 随机产生一个形体及其关键数据，并将形体加入到列表中，然后“画”出这个形体（包括显示形体的关键数据）。此后要求学习者输入面积，程序判断输入是否正确，如果不正确则要求继续输入，直到正确为止；
3. 此后程序询问学习者是否继续。如果是，则重复上一步；
4. 否则，程序进入复习模式。
   * 在复习模式下，程序“画”出学习者学过的每一个形体。此后程序结束。

继续第四次实验的编码过程，直到完成List类的编码和测试。

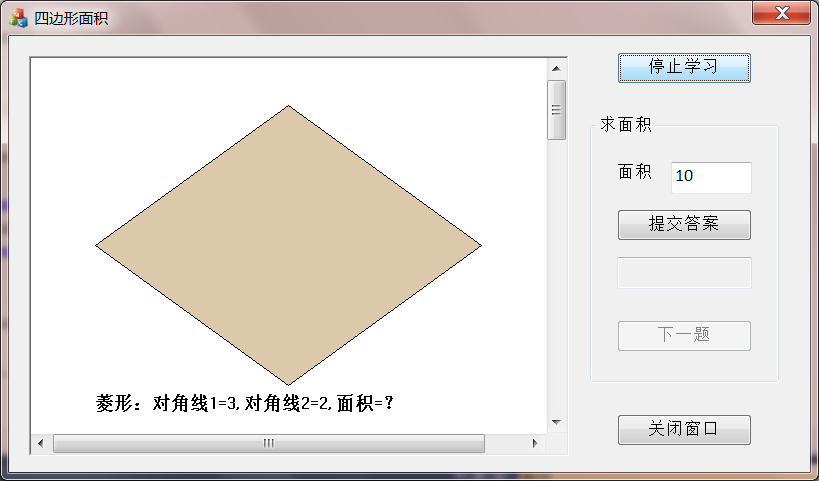
1. 将提供的压缩包解压到任一磁盘上，例如D:，将生成一个experiment文件夹；
2. 将你写好的如下文件拷贝到D:\experiment\experiment文件夹下：
   1. List.h和list.cpp
   2. 形体类的.h和.cpp文件
3. 双击D:\experiment\experiment.sln，启动该解决方案，然后将上述源文件添加到该解决方案中。具体做法是：在“解决方案资源管理器”面板中，右击解决方案的名字，然后选择“添加”->“现有项…”，在弹出的对话框中选择需要的源文件按“添加”按钮。此后在文件列表中拖拽源文件名，将其放到适当的列表项下面。
4. 打开stdafx.h文件，然后将所有你编写的头文件用#include指令添加在最后。注意头文件包含的先后顺序；
5. 保存解决方案；
6. 启动解决方案，按如下描述添加代码：
   1. 在形体类的头文件插入如下一行：

#include "Canvas.h"

* 1. 对每一种形体，删除draw()中原来的调试代码，改成如下一行：

Canvas::drawQuad(this);

* 1. 在experimentDlg.cpp模块前面的位置定义一个List类的全局对象list（必须使用这个名字），并在canvas.cpp中将该list对象说明成是外部的(extern)。在做好的代码中已经用符号对//{{和//}}标识出了你需要添加代码的地方，此后均按此操作；
  2. 在experimentDlg.cpp中编写用于列表遍历时用到的access()函数。它应该是全局的。当然，你也可以把它设计成为CexperimentDlg类的静态成员。
  3. 将随机生成形体的代码编写或粘贴在CexperimentDlg::createAquad()中。在这段代码中，关键一点是一定要将生成形体的指针加入到列表对象中；
  4. 其他需要添加代码的地方在做好的程序中用符号对//{{和//}}标识出来，请搜索这些符号以完成程序；
  5. 生成解决方案然后调试。程序运行的截图如下：



1. 界面操作流程：
   1. 按“开始学习”按钮，画出一个形体，在输入框中输入面积，然后按“提交答案”，如果正确，将显示“OK”；否则显示“答案错误”，此时重新输入直到正确为止；
   2. 按“下一题”重复学习；
   3. 按“结束学习”按钮后，所有学过的形体都会绘制在窗口中；
   4. 按“关闭窗口”退出应用。

继续完成系统的整合调试工作。

继续完成系统的整合调试工作。在完成后对代码进行Review，并考虑如何优化。最后要完成整个系统。此后需撰写实验报告并提交电子档。

注：如有学生提前完成上述工作，那么请完成选作内容。

附件1：示例代码

附件2：系统所需解决方案

附件3：选作内容指导

附件4：选作内容替换代码

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）**

Quad.h

//四边形

#ifndef QUAD\_H

#define QUAD\_H

#include<iostream>

#include"Canvas.h"

using namespace std;

class Quadrangle

{

protected:

string name;

public:

Quadrangle(string \_name="Quadrangle") :name(\_name){};

virtual double area() const = 0;

virtual void draw() const = 0;

virtual string what() const{return name;}

};

typedef Quadrangle \* QUADPTR;

#endif

Rect.h

//矩形

#ifndef RECT\_H

#define RECT\_H

#include"Quad.h"

class Rect : public Quadrangle

{

public:

Rect(int \_width = 5, int \_height = 7) :

Quadrangle("Rectangle"),width(\_width), height(\_height){}

string what()const{ return name; }

double area()const{ return width\*height; }

void draw()const{ Canvas::drawQuad(this); }

int& Width() { return width; };

int& Height(){ return height; };

private:

int width;

int height;

};

#endif

Para.h

//平行四边形

#ifndef PARA\_H

#define PARA\_H

#include"Quad.h"

#include"Canvas.h"

class Parallelogram : public Quadrangle

{

public:

Parallelogram(int \_width = 5, int \_height = 5)

:Quadrangle("Parallelogram"), width(\_width), height(\_height){}

string what()const{ return name; }

double area()const{ return width\*height; }

void draw()const{ Canvas::drawQuad(this); }

virtual int& Width();

virtual int& Height();

protected:

int width;

int height;

};

#endif

Squa.h

//正方形

#ifndef SQUA\_H

#define SQUA\_H

#include"Quad.h"

class Square: public Quadrangle

{

public:

Square(int \_SideLength = 5)

:Quadrangle("Square"), SideLength(\_SideLength){}

string what() const{ return name; }

double area() const{ return SideLength\*SideLength; }

void draw() const{ Canvas::drawQuad(this); }

int& Width() { return SideLength; };

int& Height(){ return SideLength; };

private:

int SideLength;

};

#endif

Trap.h

#ifndef TRAPEZOID\_H

#define TRAPEZOID\_H

#include"Quad.h"

class Trapezoid :public Quadrangle

{

public :

Trapezoid(int \_width=2, int \_width2=3, int \_height=7) :

Quadrangle("Trapezoid"), width(\_width), width2(\_width2), height(\_height){}

string what()const{ return name; }

double area()const{ return (width+width2)\*height/2; }

void draw()const{ Canvas::drawQuad(this); }

int& Width() { return width; };

int& Width2(){ return width2; };

int& Height(){ return height; };

private:

int width;

int width2;

int height;

};

#endif

Diam.h

//菱形\_

#ifndef DIAM\_H

#define DIAM\_H

#include"Quad.h"

class Diamond :public Quadrangle

{

public:

Diamond(int \_SideLength = 5, int \_height = 7)

:Quadrangle("Diamond"), SideLength(\_SideLength), height(\_height){}

string what()const{ return name; }

double area()const{ return SideLength\*height; }

void draw()const{ Canvas::drawQuad(this); }

int& Width() { return SideLength; };

int& Height(){ return height; };

private:

int SideLength;

int height;

};

#endif

List.h

#ifndef LIST\_H

#define LIST\_H

#include "Quad.h"

typedef void(\*ACCESSFUN)(QUADPTR);

class List

{

protected:

struct Node

{

QUADPTR quad;

Node \*next;

Node(QUADPTR q);

} \*head, \*tail;

size\_t len;

public:

List();

List(const List& s);

~List();

List& push\_back(const QUADPTR quad);

void pop\_back();

operator bool() const;

size\_t size() const;

void clear();

List& operator=(const List& l);

List& operator+=(const QUADPTR quad);

QUADPTR& operator[](size\_t index);

void traverse(ACCESSFUN f);

typedef Node\* \_range;

#endif

List.cpp

#include"stdafx.h"

List::List() : head(NULL), tail(NULL), len(0){}

List::List(const List& l) : head(l.head), tail(l.tail), len(l.len){}

List::~List(){clear();}

List::Node::Node(QUADPTR q) : quad(q){}

List& List::push\_back(const QUADPTR quad)

{

Node \*p = new Node(quad);

p->next = NULL;

if (tail == NULL) //first node

head = tail = p;

else

{

tail->next = p;

tail = p;

}

++len;

return \*this;

}

void List::pop\_back()

{

if (tail == NULL) return;

--len;

if (head == tail)

{

delete head;

head = tail = NULL;

return;

}

Node \*p = head, \*q = p->next;

while (q != tail)

{

p = p->next;q = q->next;

}

delete q;

p->next = NULL;

tail = p;

}

List::operator bool() const{return head != NULL;}

List& List::operator=(const List& l)

{

head = l.head;tail = l.tail;return \*this;

}

List& List::operator+=(const QUADPTR quad)

{

push\_back(quad);return \*this;

}

size\_t List::size() const{ return len;}

QUADPTR& List::operator[](size\_t index)

try

{

if (index >= len || len == 0) throw(0);

Node \* p = head;

size\_t i = 0;

while (i < index)

{

p = p->next;

++i;

}

return p->quad;

}

catch (int) { static QUADPTR s = NULL; return s; }

void List::clear()

{

Node \*p = head, \*q;

while (p != NULL)

{

q = p;

p = p->next;

delete q;

}

head = tail = NULL;

len = 0;

}

void List::traverse(ACCESSFUN f)

{

Node \*p = head;

while (p != NULL)

{

f(p->quad);

p = p->next;

}

}

1. **总结及心得体会：**

熟练掌握面向对象的核心知识：封装、继承、多态。只有深入理解了类和对象之间的关系，以及实际问题中各个类之间的关系才能更好地设计出相应的数据模型，对于类中的某些数据成员应当封装起来防止外界的不利因素改变本身具有的属性，通过自定义的一些特有的方法对相应的数据成员进行处理；在继承父类的属性和方法之后，由于子类自身的差异，有时需要重新定义方法或者属性，就不得不对父类的相关成员进行重载；多态又主要分为静态多态和动态多态，静态多态则主要体现在运算符重载，为了使链表的某些操作更加便于人的理解和使用，采用运算符重载的方法达到简化主函数中的代码的目的，至于动态多态，则主要体现在父类指针引用子类对象的相应应用中。

泛型编程在List链表的设计中得到了体现，链表的每一个节点的数据类型不再是我们以往常使用的基本数据类型而是我们自定义的类型，与此同时，节点的数据类型也不是固定的一成不变的类型，而是在我们自定义设计的形状类型中随机产生，在处理不同自定义类型的过程中，List链表的操作又完全一致，所以为了解决这一根本性问题，引入泛型编程，让代码具有可重用性，是程序具有普适性，也便于后期的维护。

**九、对本实验过程及方法、手段的改进建议：**

在开始一个完整的应用开发之前，最好是先建立相关模型，将实际问题抽象化，譬如数据模型，同时要梳理好各类型之间的关系，对于某些特定的功能的实现以及一些资源或者库的调用在开发之前根据需求做出详细的分析，同时也要规范整套的开发流程，落实每一步的具体任务。

对于一个完整的应用，还要对代码文件和资源文件进行规范化的管理，从而提高开发的效率。

对于基于MFC的应用开发方面的知识还是比较欠缺。之前无Windows平台的桌面图形化应用的开发经历，还是显得比较生疏。

对于迭代器的使用，譬如本次试验中List的遍历操作，还有待加强。迭代器其实本身就是泛型编程的体现。

**报告评分：**

**指导教师签字：**