1. 程序代码

程序一

# include<iostream>

using namespace std;

class Tr {

public:

Tr(int n)

{

i = n;

}

void set\_i(int n)

{

i = n;

}

int get\_i()

{

return i;

}

private:

int i;

};

void sqr\_it(Tr ob) // 对象ob作为函数sqr\_it的形参

{

ob.set\_i(ob.get\_i()\*ob.get\_i());

cout << "在函数sqr\_it内，形参对象ob的数据成员i的值为:" << ob.get\_i();

cout << endl;

}

int main()

{

Tr obj(10);

cout << "调用函数sqr\_it前, 实参对象obj的数据成员i的值为:";

cout << obj.get\_i() << endl;

sqr\_it(obj);

cout << "调用函数sqr\_it后, 实参对象obj的数据成员i的值为:";

cout << obj.get\_i() << endl;

system("pause");

return 0;

}

void sqr\_it(Tr\* ob) // 对象指针ob作为函数sqr\_it的形参

{ ob->set\_i(ob->get\_i()\*ob->get\_i());

cout<<“在函数sqr\_it内，形参对象ob的数据成员i的值为:”<<ob->get\_i();

cout<<endl;

}

void sqr\_it(Tr& ob)

{ ob.set\_i(ob.get\_i()\*ob.get\_i());

cout<<“在函数sqr\_it内，形参对象ob的数据成员i的值为:”<<ob.get\_i();

cout<<endl;

}

程序二

# include<iostream>

using namespace std;

class Ctest{

static int count; //私有成员

public:

Ctest() {

++count;cout<<"对象数量="<<count<<'\n';}

};

int Ctest::count=0; //A，对静态数据定义性说明

int main(void){

Ctest a[3];

return 0;

}

void main()

{

TStudent stu[3];//定义三个学生

stu[0].InitStudent("A");

stu[1].InitStudent("B");

stu[2].InitStudent("C");

stu[0].ExpendMoney(50);

TStudent::ShowMoney();

stu[1].ExpendMoney(98.5);

TStudent::ShowMoney();

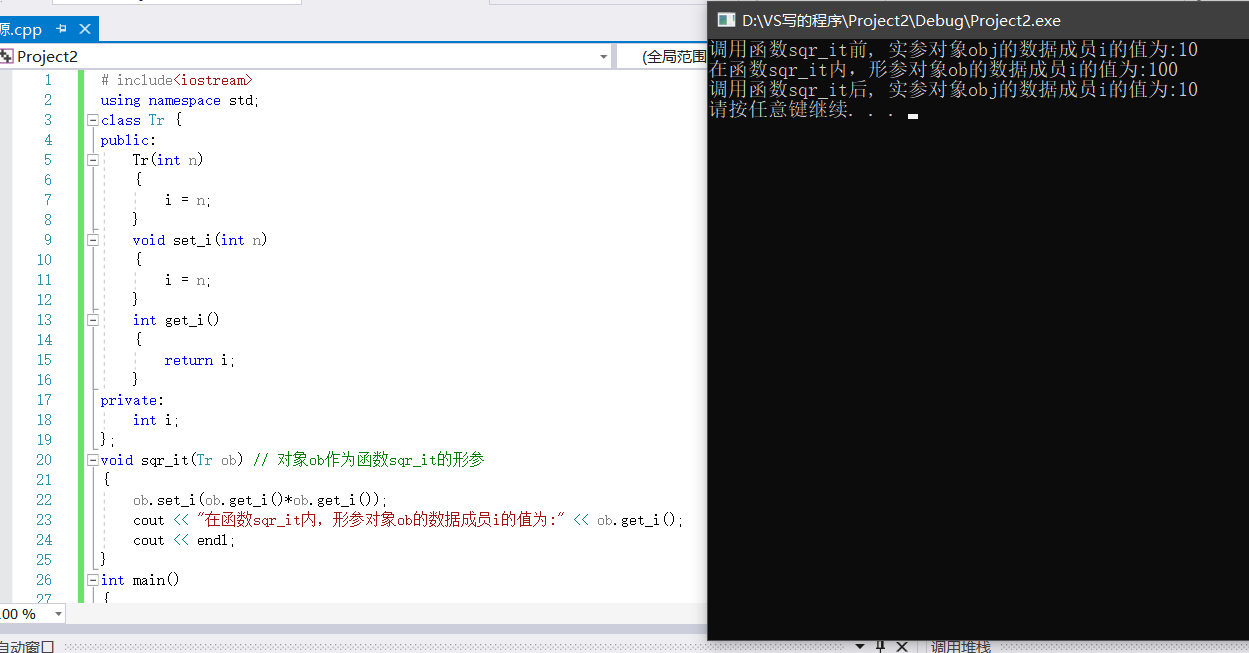
stu[2].ExpendMoney(500.53);

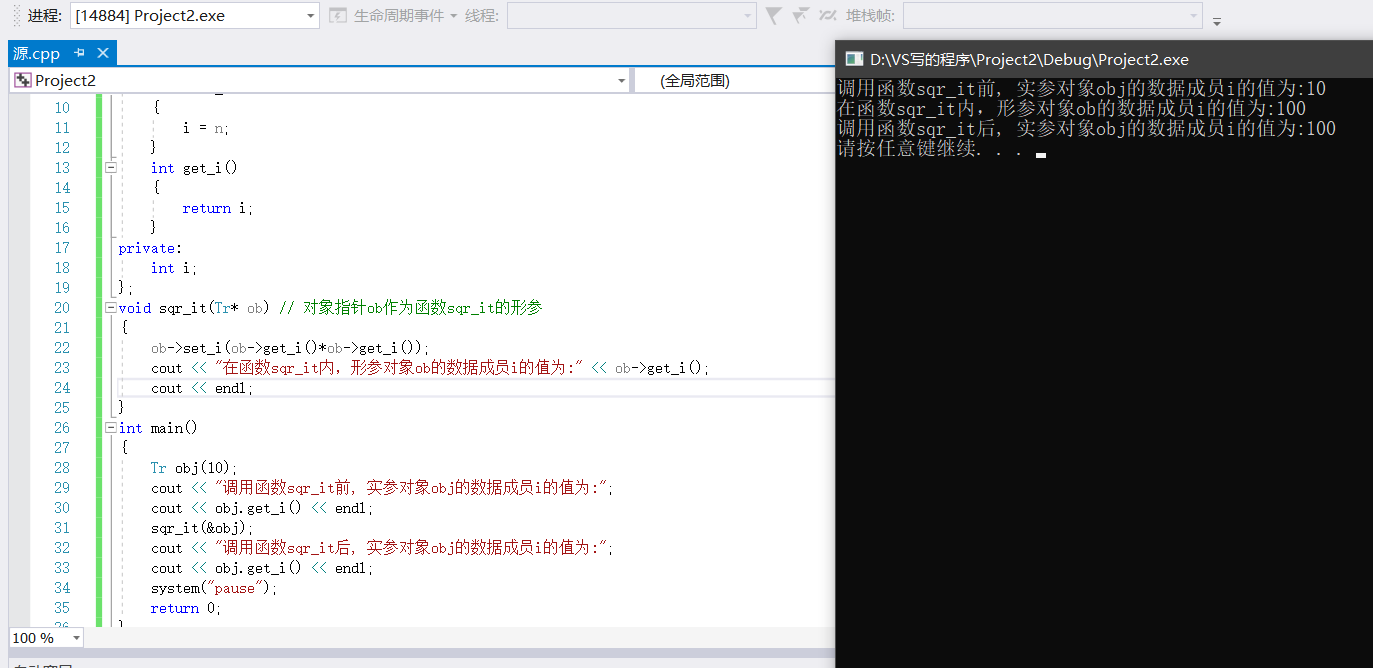
TStudent::ShowMoney();

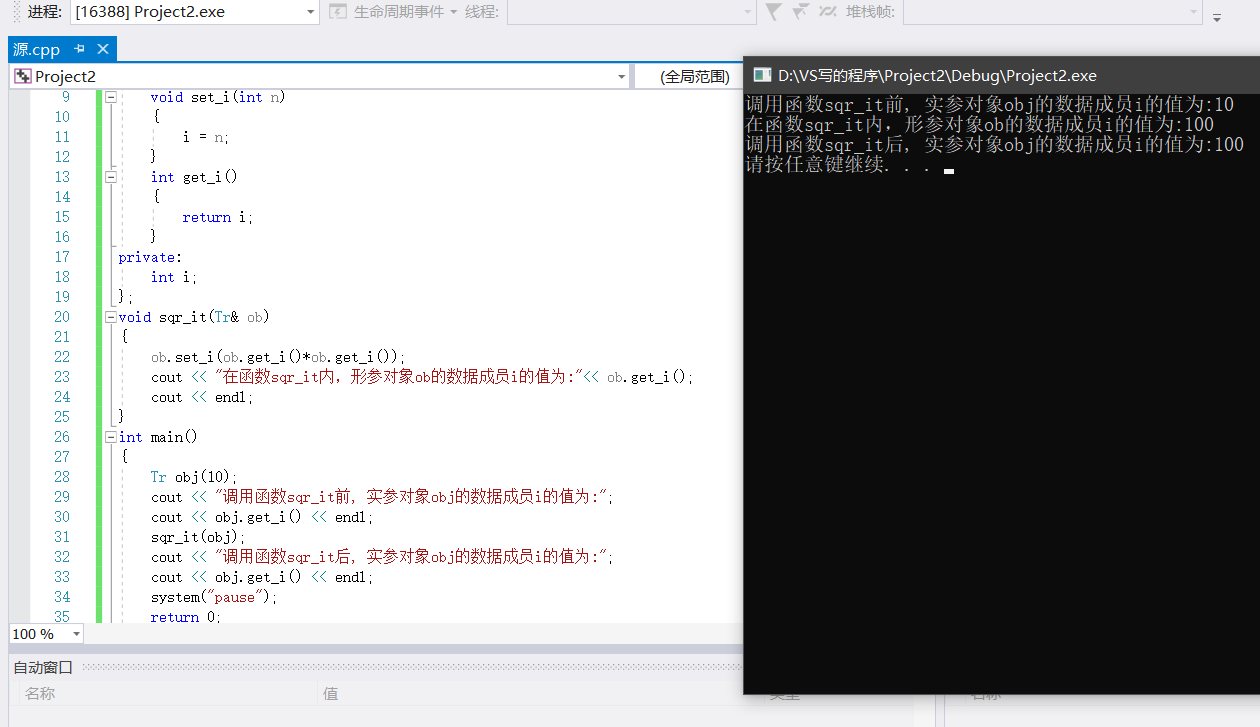
}

1. 运行结果

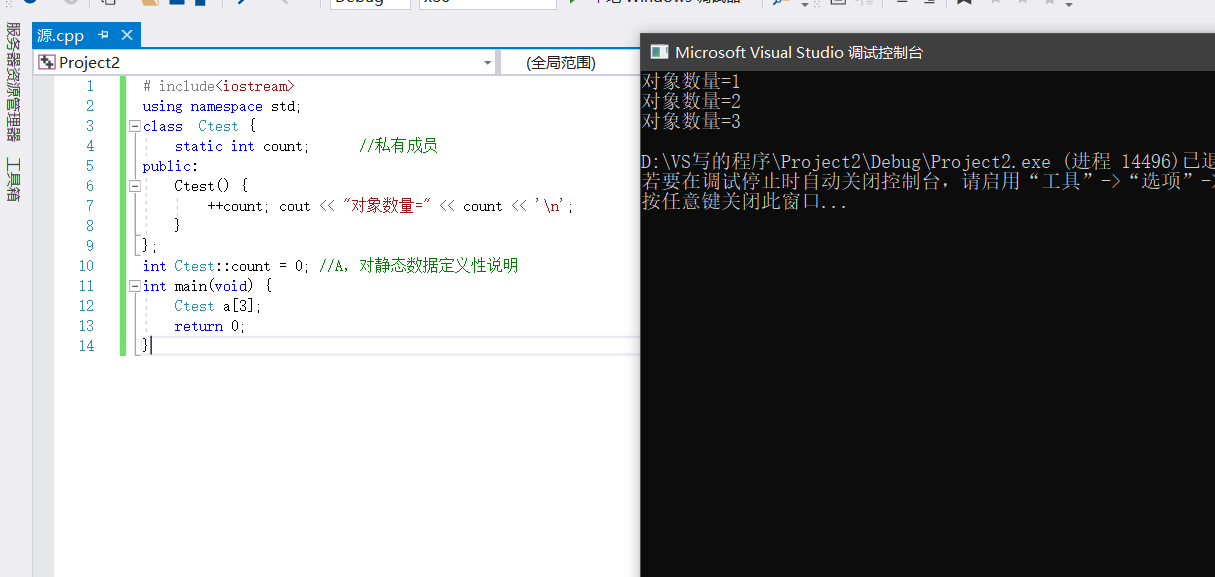
程序一

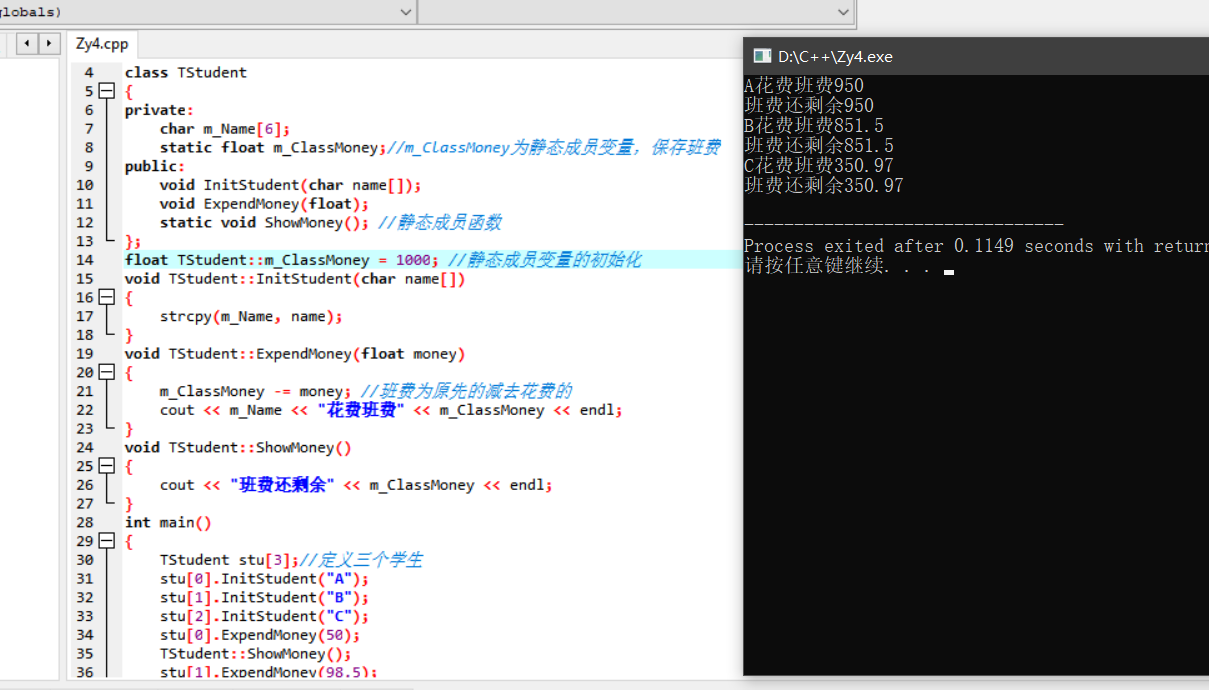






程序二





1. 心得和感想

程序一

1. 值传递：有一个形参向函数所属的栈拷贝数据的过程，如果值传递的对象是类对象或是大的结构体对象，将耗费一定的时间和空间。  
     
   2.指针传递：同样有一个形参向函数所属的栈拷贝数据的过程，但拷贝的数据是一个固定为4字节的地址。  
     
   3.引用传递：同样有上述的数据拷贝过程，但其是针对地址的，相当于为该数据所在的地址起了一个别名。  
     
   效率上讲，指针传递和引用传递比值传递效率高。一般主张使用引用传递，代码逻辑上更加紧凑、清晰。  
     
   引用传递做函数参数”是C++的特性，C语言不支持。

（1）值传递，就是把你的变量的值传递给函数的形式参数，实际就是用变量的值来新生成一个形式参数，因而在函数里对形参的改变不会影响到函数外的变量的值。  
（2）地址传递，就是把变量的地址赋给函数里形式参数的指针，使指针指向真实的变量的地址，因为对指针所指地址的内容的改变能反映到函数外，能改变函数外的变量的值。  
（3）引用传递(C++中特有，C语言没有)，实际是通过指针来实现的，能达到使用的效果如传址，可是使用方式如传值。

程序二

静态成员函数一般不访问普通数据成员，它的作用主要是访问和操作同类中的静态数据成员或全局变量。

一个类的静态成员函数与非静态成员函数不同，它不需要创建任何该类的对象就可以被调用。

Copyright ©2021-2099 Gaolei Zhao. All rights reserved