1. 程序代码

程序一

#include<iostream>

using namespace std;

class Complex {

public:

double real;

double imag;

Complex(double r = 0, double i = 0)

{

real = r; imag = i;

}

};

Complex operator+(Complex co1, Complex co2)

{

Complex temp;

temp.real = co1.real + co2.real;

temp.imag = co1.imag + co2.imag;

return temp;

}

int main()

{

Complex com1(1.1, 2.2), com2(3.3, 4.4), total1, total2;

total1 = operator+(com1, com2);

cout << "real1=" << total1.real << " " << "imag1=" << total1.imag << endl;

total2 = com1 + com2;

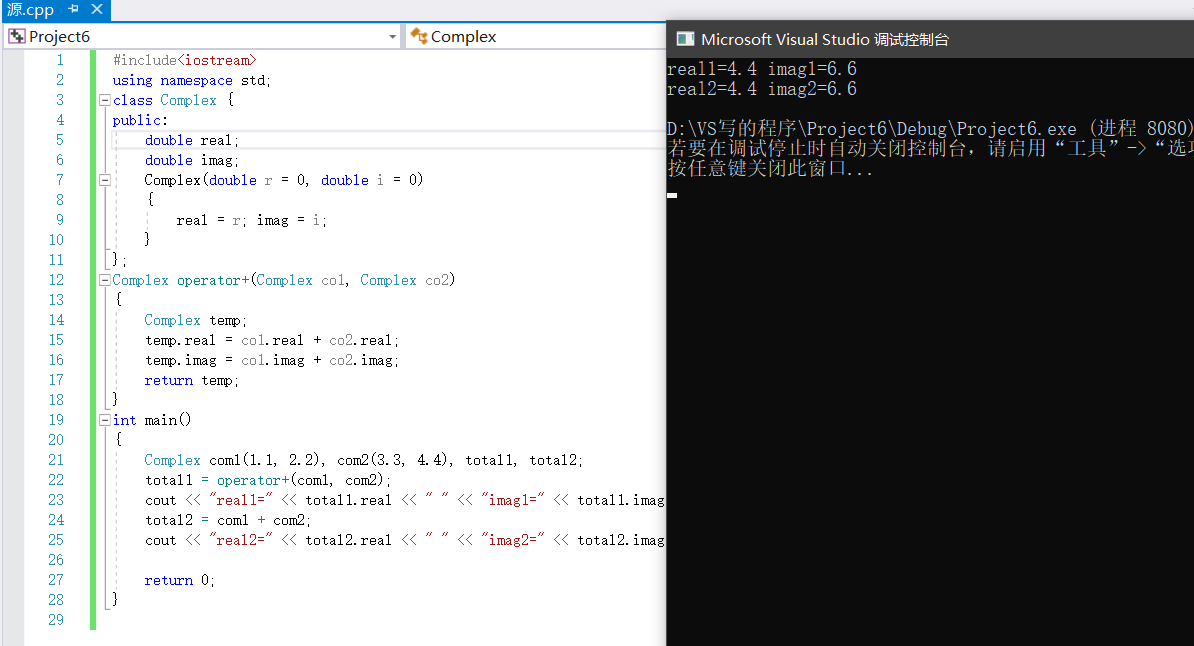
cout << "real2=" << total2.real << " " << "imag2=" << total2.imag << endl;

return 0;

}

1. 运行结果

程序一



1. 心得和感想

程序一

运算符重载是通过创建运算符函数实现的，运算符函数定义了重载的运算符将要进行的操作。运算符函数的定义与其他函数的定义类似，惟一的区别是运算符函数的函数名是由关键字operator和其后要重载的运算符符号构成的。运算符函数定义的一般格式如下：  
  
  <返回类型说明符> operator <运算符符号>(<参数表>)  
  
{  
  
     <函数体>  
  
}  
  
　2.运算符重载时要遵循以下规则：  
  
(1) 除了类属关系运算符"."、成员指针运算符".\*"、作用域运算符"::"、sizeof运算符和三目运算符"?:"以外，C++中的所有运算符都可以重载。  
  
(2) 重载运算符限制在C++语言中已有的运算符范围内的允许重载的运算符之中，不能创建新的运算符。  
  
(3) 运算符重载实质上是函数重载，因此编译程序对运算符重载的选择，遵循函数重载的选择原则。  
  
(4) 重载之后的运算符不能改变运算符的优先级和结合性，也不能改变运算符操作数的个数及语法结构。  
  
(5) 运算符重载不能改变该运算符用于内部类型对象的含义。它只能和用户自定义类型的对象一起使用，或者用于用户自定义类型的对象和内部类型的对象混合使用时。  
  
(6) 运算符重载是针对新类型数据的实际需要对原有运算符进行的适当的改造，重载的功能应当与原有功能相类似，避免没有目的地使用重载运算符。  
  
（7）重载运算符的函数不能有默认的参数，否则就改变了运算符的参数个数，与前面第3点相矛盾了；  
  
（8）重载的运算符只能是用户自定义类型，否则就不是重载而是改变了现有的C++标准数据类型的运算符的规则了，会引会天下大乱的；  
  
（9）用户自定义类的运算符一般都必须重载后方可使用，但两个例外，运算符“=”和“&”不必用户重载；  
  
（10）运算符重载可以通过成员函数的形式，也可是通过友元函数，非成员非友元的普通函数。