


## 类友元形式的运算符重载

- 由于友元函数可以访问类的私有成员和保护成员，为了方便，类非成员函数形式的运算符重载函数一般采用友元函数。
- 运算符重载为类的友元函数，需要在类内进行声明。声明类的友元运算符重载的形式为：
- friend <函数类型> operator<运算符>(<参数表>);
- 当运算符重载为类的友元函数时，由于没有隐含的this指针，因此操作数的个数没有变化，所有的操作数都必须通过函数的形参进行传递，函数的参数与操作数自左至右一一对应。

- 调用友元函数运算符的形式如下：
- $\langle \text{参数1} \rangle \langle \text{运算符} \rangle \langle \text{参数2} \rangle$
- 它等价于：
- `operator  $\langle \text{运算符} \rangle$  ( $\langle \text{参数1} \rangle$ ,  $\langle \text{参数2} \rangle$ )`
- 【例2-18】利用友元运算符重载函数实现两个复数对象的加法计算。

```
// Complex.h
class Complex
{
public:
    Complex();
    Complex(double r,double i);
    friend Complex operator+(Complex &rc1,Complex &rc2);
    void Display();
private:
    double m_real;
    double m_imag;
};
```



```
// Complex.cpp
#include "Complex.h"
#include <iostream>
using namespace std;
Complex::Complex()
{
    m_real=0;
    m_imag=0;
}
Complex::Complex(double r,double i)
{
    m_real=r;
    m_imag=i;
}

Complex operator+(Complex &rc1,Complex &rc2)
{
    Complex c;
    c.m_real=rc1.m_real+rc2.m_real;
    c.m_imag=rc1.m_imag+rc2.m_imag;
    return c;
}

void Complex::Display()
{
    cout<<"("<<m_real<<","<<m_imag
        <<"i)"<<endl;
}
```

```
// testComplex.cpp
#include "Complex.h"
#include <iostream>
using namespace std;
int main()
{
    Complex c1(1,2),c2(3,4),c3;
    c3=c1+c2;      //等价于 : c3= operator+(c1 , c2);
    cout<<"c1=";
    c1.Display ();
    cout<<"c2=";
    c2.Display ();
    cout<<"c3=c1+c2=";
    c3.Display();
    return 0;
}
```

- 在多数情况下，将运算符重载为类的成员函数和类的友元函数都是可以的，采用何种形式，可参考下面的规则：
- （1）一般情况下，单目运算符最好重载为类的成员函数；双目运算符则最好重载为类的友元函数。
- （2）以下一些双目运算符只能重载为类的成员函数：`=`、`()`、`[]`、`->`。
- （3）若一个运算符的操作需要修改对象的状态，选择重载为成员函数较好。

- ( 4 ) 运算符所需的操作数 ( 尤其是第一个操作数 ) 希望有隐式类型转换 , 则只能选用友元函数。
- ( 5 ) 当运算符函数是一个成员函数时 , 最左边的操作数 ( 或者只有最左边的操作数 ) 必须是运算符类的一个类对象 ( 或者是对该类对象的引用 ) 。如果左边的操作数必须是一个不同类的对象 , 或者是一个内部类型的对象 , 该运算符函数只能作为一个友元函数来实现。
- ( 6 ) 当需要重载运算符具有可交换性时 , 选择重载为友元函数。