面向对象方法与C++程序设计

第3章　　运算符重载

大连理工大学
主讲人-赵小薇

运算符重载方法



1 成员函数

2 友元函数

3 普通函数

重载方法

成员函数重载运算符



- 何时用成员函数重载算符?

当一元运算符的操作数，或者二元运算符的左操作数

是该类的一个对象时，重载运算符函数为成员函数

```
class Complex
{
        double dReal, dImag;
 public:
        Complex(double r, double i);
        Complex operator + (Complex &c2);
        Complex operator - ( );
        Complex operator * (double d);
        } ;
```

成员函数重载运算符



- 何时成员函数不能重载算符?

```
class Complex
{
        double dReal, dImag;
 public:
        Complex(double r, double i);
        Complex operator + (Complex &c2);
        Complex operator + (double d);
};
```

```
Complex  z ( 2.0 , 3 ) ,   k ( 3 , 4.5 ) ;
k = z + 27 ;
k = 27 + z ;
```

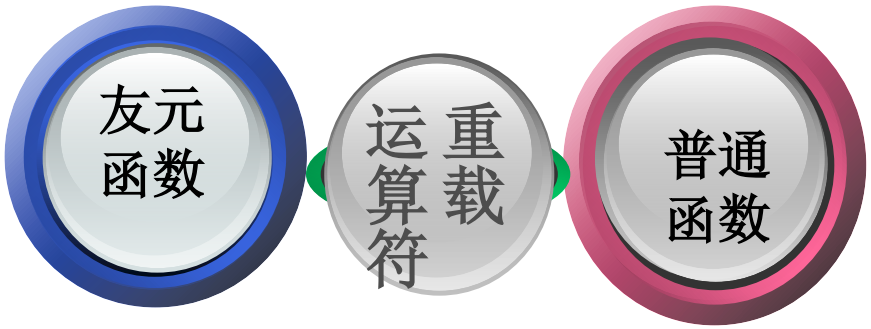
成员函数重载的“ + ” 算符；不支持交换律

友元函数重载运算符



友元函数重载运算符常用于运算符的左右操作数类型不同的情况。

```
Complex operator + (Complex &c2);   //成员函数
 Complex operator+(double d1, const Complex &c2);
friend Complex operator+( double d1, const Complex &c2);
```



成员函数与友元函数的比较



1．优先选择成员函数实现运算符重载

一般情况下单目运算符重载为类的成员函数，
尤其运算符的操作需要修改对象的状态时。
双目运算符可以重载为类的成员或者友元函数。

不能用类的友元函数重载实现的运算符



2．不能重载为成员函数的运算符

当有两个不同类型的对象进行混合运算时，若双目运算符的左操作数不是A类对象，而右操作数为A类对象，则该运算符函数不能重载为A类成员函数。

当运算符函数重载为某类的成员函数时，双目运算符的左操作数，或者单目运算符的唯一操作数，必须是该类的对象或者对象的引用。

举例



```
class Complex                                   //定义Complex类
{ public:
    Complex( ){dReal=0;dImag=0; }               //定义构造函数
    Complex(double r, double i)                 //重载构造函数
      { dReal = r;  dImag = i; }
    Complex operator + ( const Complex &c2);  //两个复数相加函数
    Complex operator + (double d);                  //复数和实数相加函数
     void print( )const;
  //友元函数重载+与-
friend Complex operator+( double d, const Complex &c) ;
friend Complex operator-( const Complex &c1, const Complex &c2) ;
friend Complex operator-( const Complex &c1) ;
private:
    double dReal; double dImag;                         //实部与虚部
};
```



```
//成员函数实现
Complex Complex::operator + (const Complex &c2){
return Complex ( dReal+c2.dReal, dImag+c2.dImag); }
Complex Complex::operator + (double  d) {
return Complex ( dReal+d, dImag);  }
void Complex::print( )const {
    cout << '(' << dReal << ", " << dImag << ')' << endl; }
//友元函数实现
Complex operator+( double d, const Complex &c) {
 return Complex ( d + c. dReal, c.dImag); }
Complex operator-( const Complex &c) {
 return Complex (-c. dReal, -c.dImag); }
Complex operator-( const Complex &c1 ,const Complex &c2) {
 return Complex (c1. dReal -c2. dReal, c1. dImag -c2.dImag); }
```



```
int main( ){
    Complex c0,c1(-3,4),c2(1,-10);
    double  d1=5.5, d2=0.5;
    d1= d1 + d2;                                  //内部定义的+操作
    c0 = c1+c2;  cout << "c1 + c2 = ";      //c1.operator +(c2)
    c0.print( );
    c0 = c1 + d1; cout << "c1 + d1 = ";     //c1.operator +(d)
    c0.print( );
    c0 = d1+ c1; cout << "d1 + c1 = ";     //operator +(d, c1)
    c0.print( );
    c0 = c1-c2; cout << "c1 - c2 = ";         //operator -(c1, c2)
    c0.print( );
    c0 = -c2;    cout << " - c2 = ";            //operator -( c2)
     c0.print( );     return 0;
}
```

运行结果



- 运行结果如下：

c1 + c2 = (-2, -6)
c1 + d1 = (3, 4)
d1 + c1 = (3, 4)
c1 - c2 = (-4, 14)
 - c2 = (-1, 10)

其他常用运算符重载



- 关系运算符　　>、 <、 ==、 >= 、<=
- 自增自减运算符　++、--
- 赋值运算符　　=
- 下标运算符　　[ ]
- 流操作运算符　 << 、>>



- 运算符重载可以像基本数据类型一样，用简洁明确的运算符操作类对象。
- 重载运算符函数可以对运算符作出新的解释，但重原有的基本语义不变。
- 运算符函数既可以重载为成员函数，也可重载义为友员函数或普通函数。
- 当一元运算符的操作数，或者二元运算符的左操作数是该类的一个对象时，以成员函数重载。
- 当一个运算符的操作需要修改类对象状态时，应该以成员函数重载。如果以成友员函数重载，可以使用引用参数修改对象。