

# 运算符重载 – 基本概念

郭 炜 刘家瑛



北京大学



# 运算符

## ■ C++预定义表示对数据的运算

- +, -, \*, /, %, ^, &, ~, !, |, =, <<, >>, != .....
- 只能用于基本的数据类型
  - 整型, 实型, 字符型, 逻辑型.....



# 自定义数据类型与运算符重载

- C++提供了数据抽象的手段:

用户自己定义数据类型 -- 类

- 调用类的成员函数 → 操作它的对象

- 类的成员函数 → 操作对象 时, 很不方便

- 在数学上, 两个复数可以直接进行+/-等运算

**Vs.** 在C++中, 直接将+或-用于复数是不允许的



# 运算符重载

- 对抽象数据类型也能够直接使用C++提供的运算符
  - 程序更简洁
  - 代码更容易理解
- 例如:
  - `complex_a`和`complex_b`是两个复数对象
  - 求两个复数的和, 希望能直接写:

`complex_a + complex_b`



# 运算符重载

## 运算符重载

- 对已有的运算符赋予多重的含义
- 使同一运算符作用于不同类型的数据时 → 不同类型的行为

## 目的

- 扩展C++中提供的运算符的适用范围, 以用于类所表示的抽象数据类型

## 同一个运算符, 对不同类型的操作数, 所发生的行为不同

- $(5, 10i) + (4, 8i) = (9, 18i)$
- $5 + 4 = 9$



# 运算符重载

- 运算符重载的实质是函数重载  
返回值类型 operator 运算符（形参表）

```
{  
  
    .....  
  
}
```



# 运算符重载

## 在程序编译时:


- 把含 运算符的表达式 → 对 运算符函数 的调用
- 把 运算符的操作数 → 运算符函数的 参数
- 运算符被多次重载时, 根据 实参的类型 决定调用哪个运算符函数
- 运算符可以被重载成普通函数
- 也可以被重载成类的成员函数



# 运算符重载为普通函数

```
class Complex {  
    public:  
        Complex( double r = 0.0, double i= 0.0 ){  
            real = r;  
            imaginary = i;  
        }  
        double real;           // real part  
        double imaginary;      // imaginary part  
};
```





```
Complex operator+ (const Complex & a, const Complex & b)
{
    return Complex( a.real+b.real, a.imaginary+b.imaginary);
} // “类名(参数表)” 就代表一个对象
```

```
Complex a(1,2), b(2,3), c;
```

```
c = a + b; // 相当于什么?
```

- 重载为普通函数时, 参数个数为运算符目数



# 运算符重载为成员函数

```
class Complex {  
    public:  
        Complex( double r= 0.0, double m = 0.0 ):  
            real(r), imaginary(m) { }    // constructor  
        Complex operator+ ( const Complex & ); // addition  
        Complex operator- ( const Complex & ); // subtraction  
    private:  
        double real;    // real part  
        double imaginary; // imaginary part  
};
```

- 重载为成员函数时, 参数个数为运算符目数减一



## // Overloaded addition operator

```
Complex Complex::operator+(const Complex & operand2) {  
    return Complex( real + operand2.real,  
                    imaginary + operand2.imaginary );  
}
```

## // Overloaded subtraction operator

```
Complex Complex::operator- (const Complex & operand2){  
    return Complex( real - operand2.real,  
                    imaginary - operand2.imaginary );  
}
```

```
int main(){  
    Complex x, y(4.3, 8.2), z(3.3, 1.1);  
    x = y + z; // 相当于什么?  
    x = y - z; // 相当于什么?  
    return 0;  
}
```