

自加/自减运算符的重载

郭 炜 刘家瑛

北京大学



自加/自减运算符的重载

- ▲ 自加 ++/自减 -- 运算符有 前置/后置 之分
- ▲ 前置运算符作为一元运算符重载
 - 重载为成员函数:
`T & operator++();`
`T & operator--();`
 - 重载为全局函数 :
`T & operator++(T &);`
`T & operator--(T &);`
- ▲ `++obj`, `obj.operator++()`, `operator++(obj)` 都调用上述函数



自加/自减运算符的重载

■ 后置运算符作为二元运算符重载

- 多写一个参数, 具体无意义
- 重载为成员函数:

`T operator++(int);`

`T operator--(int);`

- 重载为全局函数:

`T operator++(T &, int);`

`T operator--(T &, int);`

■ `obj++`, `obj.operator++(0)`, `operator++(obj,0)` 都调用上函数



```
int main(){  
    CDemo d(5);  
    cout << (d++) << ","; //等价于 d.operator++(0);  
    cout << d << ",";  
    cout << (++d) << ","; //等价于 d.operator++();  
    cout << d << endl;  
    cout << (d--) << ","; //等价于 operator--(d,0);  
    cout << d << ",";  
    cout << (--d) << ","; //等价于 operator--(d);  
    cout << d << endl;  
    return 0;  
}
```

程序输出结果:

5,6,7,7

7,6,5,5

如何编写 CDemo?



```
class CDemo {  
    private :  
        int n;  
    public:  
        CDemo(int i=0):n(i) { }  
        CDemo & operator++();    //用于前置++形式  
        CDemo operator++(int);  //用于后置++形式  
        operator int ( ) { return n; }  
        friend CDemo & operator--(CDemo &);    //用于前置--形式  
        friend CDemo operator--(CDemo &, int); //用于后置--形式  
};  
CDemo & CDemo::operator++() { //前置 ++  
    n++;  
    return * this;  
}
```



```
CDemo CDemo::operator++(int k) {    //后置 ++
    CDemo tmp(*this);    //记录修改前的对象
    n++;
    return tmp;    //返回修改前的对象
}
CDemo & operator--(CDemo & d) {    //前置--
    d.n--;
    return d;
}
CDemo operator--(CDemo & d, int) {    //后置--
    CDemo tmp(d);
    d.n --;
    return tmp;
}
```



```
operator int ( ) { return n; }
```

- int 作为一个类型强制转换运算符被重载,

```
Demo s;
```

```
(int) s ;    //等效于 s.int();
```

- 类型强制转换运算符重载时,
 - 不能写返回值类型
 - 实际上其返回值类型 -- 类型强制转换运算符代表的类型



运算符重载的注意事项

- ✦ C++不允许定义新的运算符
- ✦ 重载后运算符的含义应该符合日常习惯
 - `complex_a + complex_b`
 - `word_a > word_b`
 - `date_b = date_a + n`
- ✦ 运算符重载不改变运算符的优先级
- ✦ 以下运算符不能被重载: “.”, “.*”, “::”, “?:”, `sizeof`
- ✦ 重载运算符(), [], ->或者赋值运算符=时, 重载函数必须声明为类的成员函数