



C++  
Programming

运算符重载

Operator Overloading

2025年3月17日

学而不厌 诲人不倦

- ➡ **4.1 运算符重载的含义、方法和规则**
- ➡ **4.2 运算符重载函数作为类成员函数和友元函数**
- ➡ **4.3 重载双目和单目运算符**
- ➡ **4.4 重载流插入和流提取运算符**



## 4.1 运算符重载的含义、方法和规则

### ➤ 运算符重载的含义

与函数重载类似，对已有的运算符赋予新的含义，用一个运算符表示不同功能的运算。

**例：** << 是C++的移位运算符，又与流对象cout配合作为流插入运算符

**C++中预定义的运算符其运算对象只能是基本数据类型，而不适用于用户自定义类型（如类）**

## 4.1 运算符重载的含义、方法和规则

### ➤ 例4.1 通过成员函数实现复数的加法。

```
#include <iostream>
using namespace std;
class Complex
{ private:
double real;
double imag;
public:
Complex(){real=0;imag=0;}
Complex(double r,double i){real=r;imag=i;}
Complex complex_add(Complex &c2);
void display();
};
```

```
Complex Complex::complex_add(Complex &c2)
{
Complex c;
c.real=real+c2.real;
c.imag=imag+c2.imag;
return c;
}
void Complex::display()
{
cout<<"("<<real<<","<<imag<<"i)"<<endl;
}
```

```
int main()
{
Complex c1(3,4),c2(5,-10),c3;
c3=c1.complex_add(c2);
cout<<"c1="; c1.display();
cout<<"c2="; c2.display();
cout<<"c1+c2="; c3.display();
return 0;
}
```



## 4.1 运算符重载的含义、方法和规则

### ➤ 运算符重载的方法和规则

运算符重载的方法是定义一个重载运算符函数，在需要时系统自动调用该函数，完成相应的运算。

运算符重载实质上是函数的重载。运算符重载函数的格式是：

**数据类型 operator 运算符(形参表)**

**{ 重载处理 }**

**数据类型：**是重载函数值的数据类型。

**operator** 是保留字

C++中可以重载**除下列运算符外**的所有运算符：

**. \* :: ?:**

只能重载C++语言中已有的运算符，不可臆造新的。  
不改变原运算符的优先级和结合性。  
不能改变操作数个数。  
经重载的运算符，其操作数中至少应该有一个是自定义类型。

- ➡ 4.1 运算符重载的含义、方法和规则
- ➡ 4.2 运算符重载函数作为类成员函数和友元函数
- ➡ 4.3 重载双目和单目运算符
- ➡ 4.4 重载流插入和流提取运算符



## 4.2 运算符重载函数作为类成员函数和友元函数

### ➤ 1. 运算符重载为类成员函数

#### 声明形式

函数类型 **operator** 运算符 (形参)

{

.....

}

重载函数名是由operator和运算符联合组成。

复数加法运算符重载函数原型可以是：

**Complex operator+ (Complex &c2);**

重载为类成员函数时， 参数个数=原操作数个数-1 （后置++、--除外）

重载为友元函数时， 参数个数=原操作数个数， 且至少应该有一个自定义类型的形参。

不能重载的运算符只有5个：

.	成员运算符
.*	成员指针运算符
::	域运算符
sizeof	长度运算符
?:	条件运算符



## 4.2 运算符重载函数作为类成员函数和友元函数

### ➤ 1. 运算符重载为类成员函数

例4.2 重载运算符`+`，用于两个复数相加。

```
class Complex
{
public:
    Complex(){real=0;imag=0;}
    Complex(double r,double i){real=r;imag=i;}
    Complex operator + (Complex &c2);
    void display();
private:
    double real;
    double imag;
};
```

```
Complex Complex::operator + (Complex &c2)
{
    Complex c;
    c.real=real+c2.real;
    c.imag=imag+c2.imag;
    return c;      }

void Complex::display()
{
    cout<<"("<<real<<","<<imag<<"i)"<<endl; }
}
```

```
int main()
{Complex c1(3,4),c2(5,-10),c3;
  c3=c1+c2;
  cout<<"c1=";c1.display();
  cout<<"c2=";c2.display();
  cout<<"c1+c2=";c3.display();
  return 0;
} 编译系统将表达式c3=c1+c2 解释为 c1.operator + ( c2 )
```





## 4.2 运算符重载函数作为类成员函数和友元函数

### ➤ 2. 运算符重载为友元函数

```
#include <iostream.h>
class Complex
{public:
    Complex () {real=0;imag=0;}
    Complex (double r) {real=r;imag=0;}
    Complex (double r,double i) {real=r;imag=i;}
    friend Complex operator+ (Complex &c1, Complex &c2);
    void display();
private:
    double real;
    double imag;
};
```



## 4.2 运算符重载函数作为类成员函数和友元函数

### ➤ 2. 运算符重载为友元函数

```
Complex operator+ (Complex &c1, Complex &c2)
{ // 显式调用构造函数
  return Complex(c1.real+c2.real, c1.imag+c2.imag) ;
}
```

```
void Complex::display()
{ cout << "(" << real << ", " << imag << "i)" << endl; }
```



## 4.2 运算符重载函数作为类成员函数和友元函数

### ➤ 2. 运算符重载为友元函数

```
int main()
{   Complex c1(3,4),c2(5,-10),c3;
    c3=c1+c2;
    cout<<"c1="; c1.display();
    cout<<"c2="; c2.display();
    cout<<"c1+c2="; c3.display();
    return 0;
}
```

- ➡ 4.1 运算符重载的含义、方法和规则
- ➡ 4.2 运算符重载函数作为类成员函数和友元函数
- ➡ **4.3 重载双目和单目运算符**
- ➡ 4.4 重载流插入和流提取运算符



## 4.3 重载双目和单目运算符

### ➤ 1. 重载双目运算符

**双目的意思是运算符左边和右边的操作数均参加运算。**

如果要重载 B 为类成员函数，使之能够实现表达式 **opr1 B opr2**，其中 opr1 为 A 类对象，则 B 应被重载为 A 类的成员函数，形参类型应该是 opr2 所属的类型。经重载后，**表达式 opr1 B opr2 相当于 opr1.operator B(opr2)。**

**例4.4 定义一个字符串类String，用来处理不定长的字符串，重载相等、大于、小于关系运算符，用于两个字符串的等于、大于、小于的比较运算。**



## 4.3 重载双目和单目运算符

### ➤ 1. 重载双目运算符

```
(1) 先建立一个String类
#include <iostream.h>
#include <string.h>
// String 是用户自己指定的类名
class String
{ public:
    String(){ p=NULL; }
    String( char *str );
    void display();
private:
    char *p;
};
```

```
String::String(char *str)
{p= str;}
```

```
void String::display()
{cout<<p;}
```

```
int main()
{String string1("Hello"),string2("Book");
 string1.display();
 cout<<endl;
 string2.display();
 return 0;
}
```



## 4.3 重载双目和单目运算符

### ➤ 1. 重载双目运算符

#### (2) 重载大于运算符

```
#include <iostream.h>
```

```
#include <string.h>
```

```
class String
```

```
{ public:
```

```
    String (){ p=NULL;}
```

```
    String (char *str);
```

```
    friend bool operator>(String &string1,String &string2);
```

```
    private:
```

```
    char *p;
```

```
};
```

```
String::String(char *str)
```

```
{ p=str; }
```

```
friend bool operator <(String &string1,String &string2);  
friend bool operator==(String &string1,String &string2);
```



## 4.3 重载双目和单目运算符

### ➤ 1. 重载双目运算符

```
void String::display()  
{ cout<<p; }
```

```
bool operator>(String &string1,String &string2)  
{if(strcmp(string1.p,string2.p)>0)  
    return true;  
    else return false;  
}
```

```
int main()  
{String string1("Hello"),string2("Book");  
    cout<<(string1>string2)<<endl;  
    return 0;  
}
```





## 4.3 重载双目和单目运算符

### ➤ 2. 重载单目运算符

单目运算符只要一个操作数，由于只有一个操作数，重载函数最多只有一个参数，如果将运算符重载函数定义为成员函数还可以不用参数。

**例4.5** 有一个Time类，数据成员有时、分、秒。要求模拟秒表，每次走一秒，满60秒进位，秒又从零开始计数。满60分进位，分又从零开始计数。输出时、分和秒的值。

## 4.3 重载双目和单目运算符

### ➤ 2. 重载单目运算符

```
#include <iostream>
using namespace std;
class Time
{ public:
    Time(){hour=0;minute=0;sec=0;}
    Time(int h,int m,int s):hour(h),minute(m),sec(s){}
    Time operator++();
    void display(){cout<<hour<<":"<<minute<<":"<<sec<<endl;}
    private:
        int hour;
        int minute;
        int sec;
};
```

## 4.3 重载双目和单目运算符

### ➤ 2. 重载单目运算符

```
//前置单目运算符重载函数
Time Time::operator ++()
{
    sec++;
    if(sec>=60)
    {
        sec=sec-60;
        minute++;
        if(minute>=60)
        {
            minute=minute-60;
            hour++;
            hour=hour%24;
        }
    }
    return *this;
}
```

```
int main()
{
    Time time1(23,59,0);
    for (int i=0;i<61;i++)
    {
        ++ time1;
        time1.display();
    }
    return 0;
}
```

## 4.3 重载双目和单目运算符

### ➤ 2. 重载单目运算符

C++中除了有**前++**外，还有**后++**。同样的运算符由于操作数的位置不同，含义也不同。

**怎样区分前++和后++？**

C++给了一个方法，**在自增或自减运算符重载函数中，增加一个int 形参。**

程序员可以选择带int形参的函数做后++，也可以选择不带int形参的函数做前++。

**Time operator++();      Time operator++(int);**

## 4.3 重载双目和单目运算符

### ➤ 2. 重载单目运算符

```
Time operator++(int);
```

```
Time Time::operator++(int)
{
    Time temp(*this); // 保存修改前的对象做返回值
    ++(*this);
    return temp;
}
```

```
int main()
{
    Time time1(21, 34, 59), time2;
    cout << " time1 : ";
    time1.display();
    ++time1;
    cout << "++time1: ";
    time1.display();
    time2 = time1++;
    cout << "time1++: ";
    time1.display();
    cout << " time2 : ";
    time2.display();
    return 0;
}
```

```
time1 : 21:34:59
++time1: 21:35:0
time1++: 21:35:1
time2 : 21:35:0
```

- ➡ 4.1 运算符重载的含义、方法和规则
- ➡ 4.2 运算符重载函数作为类成员函数和友元函数
- ➡ 4.3 重载双目和单目运算符
- ➡ 4.4 重载流插入和流提取运算符



## 4.4 重载流插入和流提取运算符

### ➤ 重载流插入和流提取运算符

cin和cout分别是**istream类**和**ostream类**的对象。

C++已经对>>和<<移位运算符进行了重载，使它们分别成为流提取运算符和流插入运算符，用来输入或输出C++的标准类型数据，所以要用#include <iostream> using namespace std;把头文件包含到程序中。

**用户自定义类型的数据不能直接用<<和>>输出和输入**，如想用它们进行输入或输出，程序员必须对它们重载。

重载函数原型的格式如下：

`istream & operator >> (istream&,自定义类&);`

`ostream & operator << (ostream&,自定义类&);`

从格式上看，>>重载函数和<<重载函数只能定义为友元函数，不能定义为成员函数，因为函数有两个形参，并且第一个形参不是自定义类型。

## 4.4 重载流插入和流提取运算符

### ➤ 重载流插入和流提取运算符

```
friend ostream& operator << (ostream&,Complex&);
```

```
ostream &operator<<(ostream &output, Complex &c)
{
    output << "(" << c.real << "+" << c.imag << "i)" << endl;
    return output;
}
```

```
Complex c1(3,4),c2(5,-10),c3;
//c3=c1.complex_add(c2);
c3 = c1 + c2;
cout<<c3;
```



## Chapter 4 运算符重载

- ➡ 4.1 运算符重载的含义、方法和规则
- ➡ 4.2 运算符重载函数作为类成员函数和友元函数
- ➡ 4.3 重载双目和单目运算符
- ➡ 4.4 重载流插入和流提取运算符

### 随堂作业

1. Complex 类双目运算符重载及流插入运算符重载。
2. Time类单目运算符重载。
3. 下课前截屏或拍照上传知新平台（代码+运行结果）。



## 实验作业四 基于二维数组类的矩阵运算方法实现

1. 设计**CMatrix类**，通过**成员函数**实现矩阵**加、减、点乘、点除**运算；
2. 通过**运算符重载**实现矩阵**加、减、点乘、点除**运算；
3. 仔细设计程序界面功能；
4. 认真按格式撰写实验作业报告，补充目的、原理等各部分内容；
5. 准备演讲PPT；
6. 扩展1：查阅资料，实现**矩阵乘法、矩阵求逆**运算。
7. 扩展2：C++标准模板库**STL模板类vector**的使用

**实验作业的扩展功能不做要求，学有余力同学可以努力尝试！**

2025年3月17日-2023年4月17日



*Thank You !*

*Q & A*