

第九章 运算符重载

- 9.1 什么是运算符重载
- 9.2 运算符重载的方法
- 9.3 重载运算符的规则
- 9.4 运算符重载函数作为类成员函数和友元函数
- 9.5 重载双目运算符
- 9.6 重载单目运算符
- 9.7 重载流插入运算符和流提取运算符
- 9.8 不同类型数据间的转换



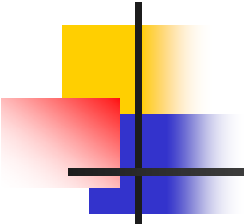
9.1 什么是运算符重载

- 重载：重新赋予新的含义
- 函数重载：对一个已有的函数赋予新的含义，使之实现新的功能。从而同一个函数名可以用来代表不同功能的函数，实现一名多用
- 运算符重载：根据用户的需要对**C++**提供的运算符赋予新的含义，使之一名多用



通过函数实现复数相加

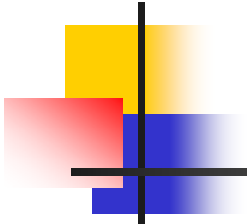
```
#include <iostream>
using namespace std;
class Complex
{public:
    Complex(){real=0;imag=0;}
    Complex(double r,double i){real=r;imag=i;}
    Complex complex_add(Complex &c2);
    void display();
private:
    double real;
    double imag;
};
```



```
Complex Complex::complex_add(Complex &c2)  
{return Complex(real+c2.real,imag+c2.imag);}
```

```
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;  
}
```



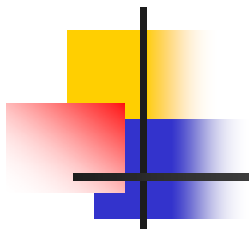
能否也和整数加法一样，直接用“+”实现复数加法运算呢？如 **$c3=c1+c2$** ;

编译系统会自动完成 **$c1$** 和 **$c2$** 两个复数相加，这就需要对运算符“+”进行重载



9.2运算符重载的方法

- 运算符重载的方法是定义一个重载运算符函数。在需要执行被重载的运算符是，系统自动调用此函数，实现相应的运算
- 运算符重载是通过定义函数来实现。运算符重载实质是函数的重载



重载运算符函数一般格式如下：

函数类型 **operator** 运算符名称（形参表列）

- **operator**是关键字，专门用于定义重载运算符函数。
- 运算符名称就是**C++**提供给用户的预定义运算符
- 函数名由**operator**和运算符组成



例子

```
#include <iostream>
using namespace std;
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; //编译系统将其解释为c1.operator+(c2)  
  cout<<"c1=";c1.display();  
  cout<<"c2=";c2.display();  
  cout<<"c1+c2=";c3.display();  
  return 0;  
}
```



运算符重载的好处

C++提供的运算符只能用于**C++**的标准数据类型数据的运算，**C++**程序设计的重要基础是类和对象，允许用户定义自己新的类型。通过运算符重载，扩大了**C++**已有运算符的作用范围，使之能用于类对象



9.3 重载运算符的规则

只能对已有的C++运算符进行重载，不允许用户定义新的运算符

不是所有的运算符都可以重载

重载不能改变运算符运算对象(即操作数)的个数.

重载不能改变运算符的优先级

重载不能改变运算符的结合性



重载运算符的函数不能有默认的参数

重载的运算符必须和用户定义的自定义类型的对象一起使用，其参数至少应有一个是类对象

用于类对象的运算符一般必须重载，但有两个例外，运算符“=”和“&”不必用户重载

应当使重载运算符的功能类似于改运算符作用于标准类型数据时所实现的功能

运算符重载函数可以是成员函数，也可以是类的友元函数，还可以是既非类的成员函数也不是友元函数的普通函数



9.4 作为类的成员函数和友元函数

- 运算符重载函数**operator+**作为**Complex**类的成员函数

Complex operator+(Complex &c2)

c1+c2 被替换为 **c1.operator+(c2)**

- 运算符重载函数**operator+**作为**Complex**类的友元函数

**Complex operator+(Complex
&c1,Complex &c2)**

c1+c2被替换为**operator+ (c1, c2)**



例子

```
#include <iostream>
using namespace std;
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;
};
```



```
Complex operator+ (Complex &c1,Complex &c2)  
{return Complex(c1.real+c2.real,  
c1.imag+c2.imag);}
```

```
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;  
}
```



两者的区别

运算符重载函数作为成员函数,必须要求运算表达式第一个参数(即运算符左侧的操作数)是一个类对象, 而且与运算符函数的类型相同。

将双目运算符重载为友元函数时, 在函数的形参表列中必须有两个参数, 不能省略, 不要求第一个参数必须为类对象。但在使用运算符表达中, 要求运算符左侧的操作数与函数第一个参数对应, 运算符右侧的操作数与函数的第二个参数对应。



Complex operator+ (Complex &c1,Complex &c2)

此函数的两个参数必须是**Complex**类对象，因此在程序中，必须给出两个**complex**对象相加

如果想将一个复数和一个整数相加，如**c1+x**（**x**为整数），则必须对以上运算符函数重载：

(1) 成员函数

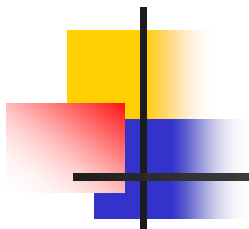
Complex::Complex operator+ (int &x)

{return Complex(real+x,imag);}

在程序中，运算符“+”左侧必须是**Complex**类的对象

应写成：**c3=c2+x;**

不能写成：**c3=x+c2** 因为**x**不是**complex**类对象



(2) 友元函数

```
Complex operator+ (int x,Complex &c)  
{return Complex(x+c.real,c.imag);}
```

表达式中必须写成 **$c3 = x + c2$** ;

不能写成 **$c3 = c2 + x$** ;

如果希望能将加法运算的交换律能适用，则应再重载一次运算符：

```
Complex operator+ (Complex &c ,int x)  
{return Complex(c.real+x,c.imag);}
```

这样 **$c2 + x$** 和 **$x + c2$** 都合法，编译系统会根据表达式的形式选择与之匹配的运算符重载函数。



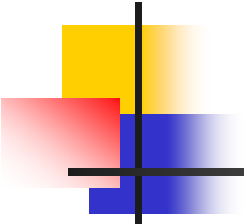
重载运算符的规定

- 有的运算符（如赋值运算符、下标运算符、函数调用运算符）必须定义为类的成员函数，有的运算符（流插入“<<”和流提取运算符“>>”、类型转换运算符）不能定义为类的成员函数
- 由于友元函数的使用会破坏类的封装，因此从原则上说，尽量把运算符函数重载为成员函数。但综合考虑，一般将单目运算符重载为成员函数，将双目运算符重载为友元函数



9.5 重载双目运算符

双目运算符（或称为二元运算符）有两个操作数，通常在运算符的左右侧。因此通常来说，重载双目运算符时，在函数中应有两个参数



定义一个字符串类，用来存放不定长的字符串，重载运算符“==”,“>”,“<”,用于字符串的等于、大于和小于的比较运算。

```
#include <iostream>
using namespace std;
class String
{public:
    String(){p=NULL;}
    String(char *str);
    void display();
private:
    char *p;
};
```

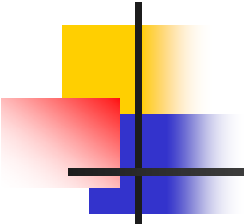
//String 是用户自己指定的类名



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

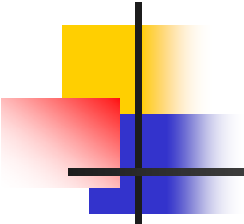
```
void String::display()  
{cout<<p;}
```

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



```
#include <iostream.h>
#include <string.h>
class String
{public:
    String(){p=NULL;}
    String(char *str);
    friend bool operator>(String &string1,String &string2);
    friend bool operator<(String &string1,String &string2);
    friend bool operator==(String &string1,String &string2);
    void display();
private:
    char *p;
};
```

增加运算符的重载，将三个运算符重载为友元函数



```
String::String(char *str)
{p=str;}
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"),
string3("Computer");
cout<<(string1>string2)<<endl;
cout<<(string1<string2)<<endl;
cout<<(string1==string2)<<endl;
return 0;
}
```




另一种写法

```
#include <iostream.h>
#include <string.h>
class String
{public:
    String(){p=NULL;}
    String(char *str);
    friend bool operator>(String &string1,String
        &string2);
    friend bool operator<(String &string1,String
        &string2);
    friend bool operator==(String &string1,String
        &string2);
    void display();
private:
    char *p;
};
```

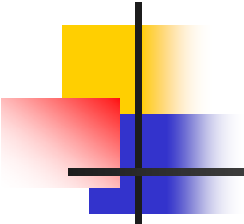


```
String::String(char *str)  
{p=str;  
}
```

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

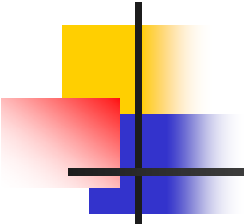
```
bool operator>(String &string1,String &string2)  
{if(strcmp(string1.p,string2.p)>0)  
    return true;  
else  
    return false;  
}
```

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



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

```
void compare(String &string1,String &string2)
{if(operator>(string1,string2)==1)
    {string1.display();cout<<">";string2.display();}
else
    if(operator<(string1,string2)==1)
        {string1.display();cout<<"<";string2.display();}
else
    if(operator==(string1,string2)==1)
        {string1.display();cout<<"=";string2.display();}
    cout<<endl;
}
```



```
int main()
{String
    string1("Hello"),string2("Book"),string3("Co
mputer"),string4("Hello");
compare(string1,string2);
compare(string2,string3);
compare(string1,string4);
return 0;
}
```



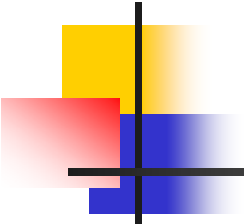
9.6 重载单目运算符

单目运算符只有一个操作数，如**!a,-b,&c,*p,++i,--i**等。由于单目运算符只有一个操作数，因此运算符重载函数只有一个参数，如果运算符重载函数作为成员函数，则还可以省略此参数。



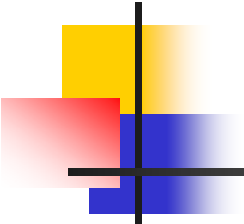
例子：自增运算符++重载

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



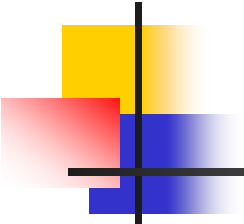
```
Time Time::operator++()  
{if(++sec>=60)  
  {sec-=60;  
    ++minute;}  
  return *this;  
}
```

```
int main()  
{Time time1(34,0);  
  for (int i=0;i<61;i++)  
    {++time1;  
      time1.display();}  
  return 0;  
}
```



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

自增（自减）运算符函数中，增加一个**int**型形参，就是后置自增（自减）运算符函数



```
Time Time::operator++()  
{if(++sec>=60)  
    {sec-=60;  
      ++minute;}  
return *this;  
}
```

```
Time Time::operator++(int)  
{Time temp(*this);  
  sec++;  
  if(sec>=60)  
    {sec-=60;  
      ++minute;}  
return temp;  
}
```



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

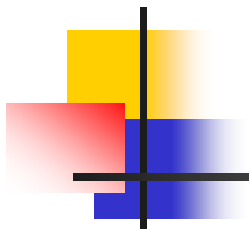
前綴自增

后綴自增



9.7 重载流运算符

- **C++**的流运算符<<和>>是**C++**类库中提供的，所有**C++**编译系统都在类库里提供输入流类**istream**和流输出类**ostream**。**cin**和**cout**分别是**istream**和**ostream**类的对象
- 在类库中提供的头文件中已经对<<和>>进行了重载，使之作为流插入运算符和流提取运算符，能用来输出和输入**C++**标准类型的数据
- 对于用户自定义的类型的的数据，不能直接用<<和>>来输出和输入。如果想用它们来输出和输入自己声明的类型的的数据，必须对它们进行重载



对<<和>>重载的函数形式:

istream & operator>>(istream &,自定义类型 &)

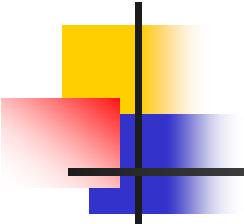
ostream & operator<<(ostream &,自定义类型 &)

只能将重载>>和<<的函数作为友元函数或普通函数，
而不能定义为成员函数



重载<<

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



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

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

//形参output是ostream类对象的引用，形参名由用户任意起。

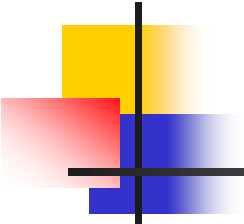
```
int main()
{Complex c1(2,4),c2(6,10),c3;
c3=c1+c2;
cout<<c3; //cout是ostream类对象，c3是complex类对象
//可解释为: operator<<(cout,c3)
return 0;
}
```



重载>>

```
#include <iostream.h>
class Complex
{public:
    friend ostream& operator << (ostream&,Complex&);
    friend istream& operator >> (istream&,Complex&);
private:
    double real;
    double imag;
};

ostream& operator << (ostream& output,Complex& c)
{output<<"("<<c.real;
 if(c.imag>=0) output<<"+";
 output<<c.imag<<"i)";
 return output;
}
```



```
istream& operator >> (istream&  
    input,Complex& c)  
{cout<<"input real part and imaginary part of  
    complex number:";  
    input>>c.real>>c.imag;  
    return input;  
}
```

```
int main()  
{Complex c1,c2;  
    cin>>c1>>c2;  
    cout<<"c1="<<c1<<endl;  
    cout<<"c2="<<c2<<endl;  
    return 0;  
}
```




9.8 不同类型数据间的转换

- 标准类型数据间的转换

(1) 隐式转换

如：

```
int i=6;
```

```
i=7.5+i;
```

(2) 显示转换

类型名（数据）

```
int (89.5)
```



用转换构造函数进行类型转化

以**Complex**类为例来看几种构造函数

默认构造函数 **Complex** () ;

用于初始化的构造函数**Complex** (**double r, double i**) ;

用于复制对象的复制构造函数 **Complex** (**Complex &c**) ;

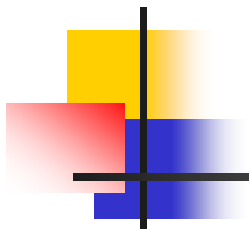
转换构造函数**Complex** (**double r**) { **real=r;**
image=0; }



转换构造函数（**conversion constructor function**）也是一种构造函数，其作用是将一个其他类型的数据转换成一个类的对象，只能有一个参数。

使用转换构造函数将一个指定的数据转化为类对象的方法：

- 声明一个类
- 在类中定义一个只有一个参数的构造函数，参数的类型是需要转化的类型，在函数体中指定转换的方法
- 在该类的作用域内用**类名**将指定数据类型的数据转换为此类对象



用转换构造函数可以将一个指定类型的数据转换为类的对象。但不能将一个类的对象转换为一个其他类型的数据

C++提供类型转换函数(**type conversion function**)，其作用是将一个类的对象转换为另一类型的数据



类型转换函数的一般形式:

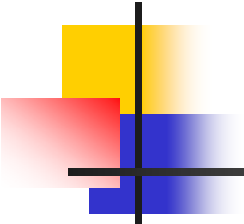
operator 类型名 ()

{实现转换的语句}

如: **operator double** () {return real; }

函数返回**double**型变量**real**的值，作用是将一个
Complex类对象转换为一个**double**型数据

在函数名前面不能指定函数类型，函数没有参数；其
返回值类型有函数名中的类型名来确定；类型转换函
数只能作为成员函数，因为转换的主体是本类的对象；
不能作为友元函数或普通函数



```
#include <iostream>
using namespace std;
class Complex
{public:
    Complex(){real=0;imag=0;}
    Complex(double r,double i){real=r;imag=i;}
    operator double() {return real;} //类型转换函数
private:
    double real;
    double imag;
};

int main()
{Complex c1(3,4),c2(5,-10),c3;
  double d;
  d=2.5+c1; //要求将一个double数据与Complex类数据相加
  cout<<d<<endl;
  return 0;
}
```