C++语言程序设计
第四章 运算符重载

王焦乐

http://faculty.hitsz.edu.cn/jlwang

哈尔滨工业大学（深圳）

机电工程与自动化学院

本章主要内容

- 为什么要对运算符重载
- 对运算符重载的方法
- 运算符重载的规则
- 运算符重载函数作为类成员函数和友元函数
- 重载双目运算符
- 重载单目运算符
- 重载流插入运算符和流提取运算符
- 有关运算符重载的归纳
- 不同类型数据间的转换

0、为什么要对运算符重载

- 重载（overloading）：“一名多用”
- 运算符重载（operator overloading）：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，这就是运算符重载。**
- 例子：
  - <<，位移运算符（左移）；cout<<，流插入运算符
- 定义一个重载运算符的函数，本质上是函数重载
- 运算符重载使C++具有更好的扩充性和适应性

0、为什么要对运算符重载

```
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.image;
  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;
}
```

能否用+运算符实现复数加法?

1、对运算符重载的方法

- 运算符重载函数的格式是：

  函数类型 operator 运算符名称(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

- 两种形式：重载为类成员函数；重载为友元函数。

- 复数例子

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

1、对运算符重载的方法

```
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;
};
```

运算符重载函数定义

```
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;
}
```

运算符重载函数调用

重载为成员函数的解释

（1）用运行符重载函数取代了例 4.1中的加法成员函数，从外观上看函数体和函数返回值都是相同的。

（2）在主函数中的表达式c3=c2+c1 取代了例4.1中的c3=c1.complex_add(c2)，编译系统将表达式c3=c1+c2 解释为

c1.operator+(c2)

对象c1调用的重载函数operator+ ，以c2为实参计算两个复数之和。

一个复数和一个整数相加

请考虑在例4.2中能否用一个常量和一个复数相加？如

```
c3 = 3 + c2;  // 错误
```

应该定义对象：**Complex c1(3.0,0)**：

```
c3 = c1 + c2;
```

注意：运算符重载后，其原来的功能仍然保留，编译系统根据运算表达式的上下文决定是否调用运算符重载函数。

运算符重载和类结合起来，可以在C++ 中定义使用方便的新数据类型

2、运算符重载的规则

（1）C++只允许已有的部分运算符实施重载。

类型	操作
双目算术运算符	+，-，*，/，%
关系运算符	==,!=,<,>,<=,>=
逻辑运算符	||,&&,!
单目运算符	+,-,*,&
自增自减	++,--
位运算符	|,&,~,^,<<,>>
赋值运算符	=,+=,-=,*=,/=,%=,|=,^=,<<=,>>=
动态空间分配	new,delete,new[],delete[]
其他	(),->,->*,,,[]

（2）不能重载的运算符有五个：

:: (scope resolution), **.** (member access), sizeof(query size)
.* (member access through pointer to member), **?:** (ternary conditional)

2、运算符重载的规则

（3）重载不改变操作数的个数。

（4）重载不改变运算符的优先级与结合性。

（5）运算符重载函数不能带默认值参数。

（6）运算符重载函数必须与自定义类型的对象联合使用，其参数至少有一个类对象或类对象引用。

（7）C++默认提供 = 和 & 运算符重载

（8）运算符重载函数可以是类成员函数也可以是类的友元函数，还可以是普通函数。

3、运算符重载函数作为类成员函数和友元函数

- 运算符重载函数两种形式：重载为类成员函数；重载为友元函数。
- 重载为类成员函数可以少一个函数的参数
- =，[]，()，->必须为成员函数
- 流插入<<和流提取运算符>>，类型转换运算符必须为友元函数
- 一般将单目和复合运算符重载为成员函数
- 一般将双目运算符重载为友元函数

3、运算符重载函数作为类成员函数和友元函数

□ 双目运算符重载为友元函数

```
class Complex
{
  public:
    Complex(){real=0;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 &c2);
```

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

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

重载为友元函数c1+c2的解释

加法运算符重载为友元函数，C++ 在编译时将表达式c1+c2解释为

operator+(c1, c2)

即相当于执行以下函数

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

普通函数是不能直接访问对象的私有成员，如果普通函数必须访问对象的私有成员，可调用类的公有成员函数访问对象的私有成员。这会降低效率。

一个复数和一个整数相加

如想将一个复数和一个整数相加，运算符重载函数作为**成员函数**定义如下：

```
Complex Complex::operator+( int & i )
{  return Complex( real + i , imag ) ; }
```

注意在运算符**+的左侧必须是Complex类对象**，程序中可以写成：

```
c3 = c2 + n
```

不能写成： `c3 = n + c2`

所以，重载为**成员函数无法实现加法交换律**

一个复数和一个整数相加

将一个复数和一个整数相加，将**运算符重载函数定义为友元函数**：

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

友元函数不要求第一个参数必须是类类型，但是要求实参要与形参一一对应：

```
c3 = n + c2        //   顺序正确
c3 = c2 +n         //   顺序错误
```

一个复数和一个整数相加

为了实现加法的交换率，必须定义两个运算符重载函数，可以用下面两个组合中任意一个：

（1）成员函数（左操作数是对象，右操作数是非对象）、友元函数（左操作数是非对象，右操作数是对象）

```
Complex Complex::operator+(int &a){…}
```

```
friend Complex operator+(const int &a1, const Complex &c2);
```

（2）友元函数（左操作数是对象，右操作数是非对象）、友元函数（左操作数是非对象，右操作数是对象）

```
friend Complex operator+(const Complex &c1, const int &a2);
```

```
friend Complex operator+(const int &a1, const Complex &c2);
```

4、重载双目运算符

- 双目运算符有两个操作数，对应重载函数两个参数
- 例子4.4：

声明一个字符串类String，用来存放不定长字符串。重载“==”，“<”，“>”，使他们能用两个字符串的比较运算。

（1）先建立一个String类，搭建一个最简单的框架

```
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、重载双目运算符

（2）再增加其他必要内容，如重载运算符“>”

```
class String
{
public:
  String(){ p=NULL; }
  String( char *str ) ;
  friend bool operator>(String &str1,String &str2);
  void display();
private:
  char *p;
};
```

运算符重载函数为友元函数

```
bool operator>(String &str1, String &str2)
{
  if(strcmp(str1.p,str2.p)>0) return true;
  else return false;
}
```

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

4、重载双目运算符

（3）扩展到3个运算符

```
friend bool operator>(String &str1,String &str2);
friend bool operator<(String &str1,String &str2);
friend bool operator==(String &str1,String &str2);
```

```
bool operator>(String &str1, String &str2)
{
  if(strcmp(str1.p,str2.p)>0) return true;
  else return false;
}
bool operator<(String &str1, String &str2)
{
  if(strcmp(str1.p,str2.p)<0) return true;
  else return false;
}
bool operator==(String &str1, String &str2)
{
  if(strcmp(str1.p,str2.p)=0) return true;
  else return false;
}
```

```
int main()
{
  String string1("Hello"),
         string2("Book"),
         string3("Computer");
  cout<<(string1>string2)<<endl;
  cout<<(string1<string3)<<endl;
  cout<<(string1==string2)<<endl;
}
```

4、重载双目运算符

(4) 进一步完善，使输出更加直观

```
void compare(String &str1,String &str2)
{
  if(operator>(str1,str2)==1)
  {str1.display();cout<<">";str2.display();}
  else if(operator==(str1,str2)==1)
  {str1.display();cout<<"=";str2.display();}
  else if(operator<(str1,str2)==1)
  {str1.display();cout<<"<";str2.display();}
  cout<<endl;
}
```

```
int main()
{
  String string1("Hello"),
         string2("Book"),
         string3("Computer"),
         string4("Hello");
  compare(string1,string2);
  compare(string2,string3);
  compare(string1,string4);
}
```

指导思想：

先搭框架，逐步扩充，由简到繁，最后完善。

边编程，边调试，边扩充。

5、重载单目运算符

- 单目运算符有一个操作数(!a, &c,++i)，对应重载函数一个参数
- 例子4.5：有一个Time类，数据成员有分、秒。要求模拟秒表，每次走一秒，满60秒，秒又从零开始计数，分钟数加1。

```
class Time
{
public:
  Time(){minute=0;sec=0;}
  Time(int m,int s):minute(m),sec(s){}
  Time operator++();    //前置++重载函数
  Time operator++(int);//後置++重载函数
  void display()
  {cout<<minute<<":"<<sec<<endl;}
private:
  int minute;
  int sec;
};
```

5、重载单目运算符

```
Time Time::operator++(){
  if(++sec>59)
  {
    sec-=60;
    ++minute;
  }
  return *this;
}
```

```
Time Time::operator++(int){
  Time temp(*this);
  if(++sec>59)
  {
    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();
}
```

6、重载流插入运算符和流提取运算符

- 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，必须对它们重载。
- >>重载函数和<<重载函数只能定义为友元函数，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&);

6、重载流插入运算符和流提取运算符

□ 用<<，>>重载函数输出，输入复数。

在类中声明<<，>>重载函数是友元函数

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

在类外定义友元函数：

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

6、重载流插入运算符和流提取运算符

```
class Complex
{
  public:
    Complex(double r=0,double i=0){real=r;imag=i;}
    friend Complex operator+(Complex &c1,Complex &c2);
    friend ostream& operator<<(ostream &output, Complex &c);
    friend istream& operator>>(istream &input, Complex &c);
  private:
    double real;
    double imag;
};
```

```
ostream& operator<<(ostream &output, Complex &c)
{
  output<<"("<<c.real<<"+"<<c.imag<<")";
  return output;
}
istream& operator>>(istream &input, Complex &c)
{
  cout<<"input real part: ";
  input>>c.real;
  cout<<"input imaginary part: ";
  input>>c.imag;
  return input;
}
```

```
int main()
{
  Complex c1,c2,c3;
  cin>>c1>>c2;
  c3=c1+c2;
  cout<<"c1="<<c1<<endl;
  cout<<"c2="<<c2<<endl;
  cout<<"c3="<<c3<<endl;
}
```

分析C++怎样处理“cout<<c3;”语句：

运算符的**左边是ostream的对象cout**，右边是程序员自定义类**complex的对象c3**，语句符合运算符重载**友元函数**operator<<的形参类型要求，系统调用友元函数，C++把这个语句解释为：

operator<<(cout , c3);

通过形参引用传递，函数中的output就是cout，函数中的c就是c3，函数就变成：

```
{ cout <<"("<<c3.real<<"+"<<c3.imag<<"i)"<<endl;
 return cout; }
```

return **cout** 是将输出流现状返回。C++ 规定**运算符<<重载函数**第一个参数和函数的类型必须是 ostream 类型的引用，目的是为了**返回cout的当前值，以便连续输出。**

7、有关运算符重载的归纳

(1) 运算符重载使程序易于理解、易于操作，使主程序简单易读。

(2) 具体做法：

①确定运算符+类

②设计重载函数（友元或成员），函数功能由设计者指定

③实际工作中可以通过头文件的函数接口使用

④了解函数原型

(3) 运算符重载中使用引用（reference）

(1) 用作形参，不生成实参副本，减少时间、空间开销

(2) 用作返回值，可以出现在=左侧，作为左值（left value）

8、不同类型数据间的转换

- 标准类型数据间的转换

  类型名(数据)；(类型名)数据；

```
int a=6;
a=7.5+a; //隐式转换
int(89.5);(int)89.5; //显式转换
```

- 其他类型数据转换为类对象

  类名(指定类型的数据)

- **转换构造函数：只有一个形参**

  Complex(double r) {real=r; imag=0;}

```
c2=c1+2.5;
c2=c1+Complex(2.5);
```

- 默认构造函数

  Complex();

- 用于初始化的构造函数

  Complex(double r=0, double i=0);

- 复制构造函数

  Complex(Complex &c);

8、不同类型数据间的转换

- 类对象转换为其他类型数据
- 类型转换函数（type conversion function）

operator 类型名（）
{实现转换的语句}

```
operator double ()
{return real;}
```

```
double d;
d=2.5+c1; //c1转成double，再相加返回double
c3=c1+c2; //两个Complex相加，返回Complex
d=c1+c2;  //两个Complex相加，转成double后返回double
```

- 如果使用类型转换函数，必须删掉运算符+重载函数

```
public:
  Complex(){real=0;imag=0;}
  Complex(double r,double i):real(r),imag(i){};
  Complex(double r){real=r;imag=0;}
  friend Complex operator + (Complex &c1, Complex &c2);
  operator double (){return real;}
```

```
c2=c1+2.5;  //错误，产生二义性
```

小结

- 为什么要对运算符重载
- 对运算符重载的方法
- 运算符重载的规则
- 运算符重载函数作为类成员函数和友元函数
- 重载双目运算符
- 重载单目运算符
- 重载流插入运算符和流提取运算符
- 有关运算符重载的归纳
- 不同类型数据间的转换