运算符重载

C++程序设计

徐延宁 xyn@sdu.edu.cn
数字媒体技术教育部工程研究中心
山东大学软件学院

运算符重载

基本概念与应用

- 函数**重载**是指完成不同功能的函数可以具有相同的函数名，不同的参数个数，或不同的参数类型。仅返回值不同时，不能定义为重载函数。
- 编译器根据函数的实参来确定应该调用哪一个函数。

```
int  fun(int a,  int b)
{   return a+b;  }
```

```
int  fun (int a)
{   return a*a;  }
```

```
void  main(void)
{  cout<<fun(3,5)<<endl;
   cout<<fun(5)<<endl;
}
```

8

25

运算符重载

基本概念与应用

- 运算符重载：为运算符（+-*/><...）定义一个（若干个）（**运算符重载**）**函数**，运算时执行对应的函数
  - <返回类型> **operator <运算符>**(<参数表>)
  - operator是运算符重载函数关键字，
  - **operator <运算符>**是函数名
  - 运算符的**操作数**映射为重载函数的参数

```
Vector v1, v2, v3;
v1 = v2 + v3;
v1 = v2.operator+(v3)
```

```
Class Vector{
    Vector operator + (Vector &);
}
```

- 运算符重载函数通常是
  - 类的**成员函数**
  - 或者是**友元函数**。
- 函数参数是运算符的操作数

```
A a1, a2;
```

```
a1 + a2
a1.add(a2)⇔ a1.operator+(a2)
add(a1,a2) ⇔operator+(a1 , a2)
```

```
a1 == a2
a1. operator==(a2)
operator== (a1 , a2)
```

```
a1 > a2
a1.operator>(a2)
operator>(a1,a2)
```

运算符重载

主要内容

- (需要你掌握的)重载的运算符
  - 双目运算符 + - * / > < +=
  - 单目运算符 ++ -- !
  - 流运算符 >> <<
  - 赋值运算符 =
  - 类型转换函数
  - 其他运算符（不常用optional）
    - 构造函数，例如，int Integer
    - []()等等
- 对应的重载函数名
  - operator+ - * / > < +=
  - operator ++ -- !
  - operator >> <<
  - operator =
  - operator int /double

建议百度，complex 运算符重载，
读懂、写出complex运算符重载函数

5

一个简单的例子

```
void main(void)
{Vector r1(3,2),r2(1,4), r3,r4;
    r1.Show (); r2.Show ();
    r3=r1+r2;
    r3.Show ();
    r4=r1+r2+r3;
    r4.Show ();
}
```

```
class Vector{
        float i;  float j;
public:
  Vector(float a=0.0,float b=0.0){   i=a;    j=b; }
   void Show(void){cout<<“i="<<i<<'\t’<<“j="<<j<<endl;}
   Vector operator+(Vector &);//重载运算符+
};

Vector Vector::operator + (Vector &r)
{   return Vector(i+r.i, j+r.j);
}
```

运算符重载

- 运算符重载的意义：
  - 使用角度：运算符重载对用户更加友好，使用便捷
- 缺点：
  - 实现稍微困难；用途有限，有时破坏程序的可读性
- 运算符重载不是必须的，Java没有运算符重载机制，不影响其逻辑表达能力

```
class A
{       int i;
public:
        A(int a=0):i(a){ }
        void Show(void){ cout<<"i="<<i<<endl;  }
        A AddA(const A &a)//函数实现加法
        { A t;          t.i=i + a.i;      return t;       }
        A operator +(const A &a)    //重载运算符+
        { A  t;         t.i=i+a.i;        return t;       }
};
void main(void)
{       A a1(10),a2(20),a3;
        a3=a1 + a2;   //可读性相对好
        a3=a1.AddA(a2);     //略微抽象
        a3.Show ();
}
```

运算符重载

+ 和 +=

```
class A
{       int i;
        public:A(int a=0){ i=a;}
        A operator +(const A &a)      //重载运算符+
        {       A  t;   t.i=i+a.i;        return t;        }
        void operator+=(const A &a)
        {       i=i+a.i;}
};
void main(void)
{       A a1(10),a2(20),a3;
        a3=a1+a2; //相当于a3=a1.operator+(a2)或者 a3 = operator+(a1,a2)
        a1+=a2; //相当于a1.operator+=(a2) 或者 operator+=(a1,a2)
}
```

+=重载函数，其返回值类型为void。

因为+= -= *=等修正运算，不会产生新的对象

8

运算符重载

补充说明-1

- 运算符重载为成员函数的参数只能有二种情况：没有参数或带有一个参数。
  - 对于单目运算符(如++)，重载时通常不能有参数；
  - 对于双目运算符，只能带有一个参数。参数可以是对象，对象的引用，或其它类型的参数。
- 只能对C++中已定义了的运算符进行重载。
- 重载不能改变运算符的优先级和结合律。
- 重载不能改变运算符运算对象(即操作数)的个数
- 重载不应该反常识，功能应当类似于其作用于标准类型数据时的功能，以增强程序**可读性**，而不应该是**脑筋急转弯**。
  - vec1 + vec2　dog1 + dog2　　dog1 + cat1

9

补充说明-2

- 有些运算符不允许重载
  - 1 . (点运算符)s.length，
  - 2 域运算符，Math::PI
  - 3 ?: (条件运算符);
  - 4 sizeof 不可以重载
- **禁止**重载（修改）基本类型数据的运算符。
  - 3 + 3(不能重载);
  - point1+point2 ，point1 + 3， 可以重载
- 运算符重载函数不能是静态成员函数
  - a.operator+(b)，operator+必然是属于对象a的函数，而不是类函数（静态函数）

10

运算符重载

成员函数 vs 友元函数

- 运算符重载为类成员函数，成员函数所属对象是一个操作数，另一个操作数是函数参数

A a ,b , c;

c=a+b; 实际上是c=a.operator+(b); 有参有返回值

c=++a; 实际上是c=a.operator++( ); 无参有返回值

c+=a; 实际上是c.operator+=(a); 有参无返回值

- 运算符重载为类的友元函数(或者普通函数)，参与运算的对象全部成为函数参数。

c=a+b;	实际上是 c=operator+(a, b);两个参数	**friend A operator+ (A &a, A &b)**
c=++a;	实际上是 c=operator++(a); 一个参数	**friend A operator++ (A &a)**
c+=a;	实际上是 operator+=(c, a);	**friend void operator+= (A &c, A &a)**

形参前面，自行添加const，养成好习惯

11

运算符重载

成员函数 vs 友元函数

```
class Complexx{
    double real;    double imag;
public:
    Complexx(double r = 0.0, double i = 0.0)   {
        real = r;         imag = i;    }
    friend Complexx operator+(Complexx , Complexx );
    Complexx operator-(Complexx &);
    void display();
};
```

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

12

成员函数 vs 友元函数

- 如果参数有基本数据类型时，则运算符建议重载为友元函数。例如：
  - Complex Complex::operator+(int &i) {return Complex(real+i,imag);}
  - 则限制使用者必须写作 c3=c2+100;
- 如果需要写成**c3=100+c2;** 则只能重载为友元函数。100不是类对象，100.operator(c2) 编译错误

```
friend Complexx operator+(Complexx , double );      c1 = c2 + 1;
friend Complexx operator+(double,  Complexx);       c1 = 1+c2;
friend Complexx operator+(Complexx,  Complexx);     c1 = c2+c3;
```

复数的加法，可能对应三个运算符重载函数

成员函数 vs 友元函数

- **双目运算符**一般（习惯上）重载为**友元函数**。
  - 例如：a + b，a==b, a && b ,
- **单目运算符（++ ！）和复合运算符（+=）**一般（习惯上）重载为**成员函数**。
  - 例如：a++; a+=b
- **流输出运算符“<<”、流输入运算符“>>”**、类型转换运算符**只能作为友元函数**。
- **赋值运算符=**、下标运算符[]、函数调用运算符()、成员运算符-> 只能定义为类的**成员函数**。

自加运算符的重载

- 虽然运算后对象a的值一致，但先自加或后自加的函数的返回值不一致，必须在重载时予以区分。
  - A a, b;

b=++a; **b = a.operator++() => A operator ++( ){ .... }**

b=a++; **b = a.operator++() =>A operator ++(int){ .... }**

参数的存在只是为了区分是前置还是后置形式

```
class A{int i;
public:
        A(int a=0)      { i=a;   }
        A operator++();
        A operator++(int n);
};
void main(void){
        A a1(10),a2,a3;
        a2=++a1;        a3=a1++;
}
```

```
A A::operator++(){i++; return *this;}
```

```
A A::operator++( int n){
        A  t;       t.i=i;       i++;
        return t;
}
```

自加运算符的重载

- 完整例子：Time；要求-信手拈来

```
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;
}
```

```
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;
};
```

运行结果如下：

```
time1 : 34:59      (time1原值)
++time1: 35:0      (执行++time1后time1的值)
time1++: 35:1      (再执行time1++后time1的值)
time2 : 35:0       (time2保存的是执行time1++前time1的值)
```

```
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;
}
```

运算符重载

流运算符的重载– “<<”输出复数

```
#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);
    friend ostream& operator << (ostream&,Complex&);
private:
   double real;
   double imag;
};
```

```
int main( ){
      Complex c1(2, 4), c2(6, 10), c3;
      c3=c1+c2;
      cout<<c3;  // operator<<(cout,c3)
      cout << c1 << c2;
// (cout<<c1) << c2  =>  cout << c2 // 拼合了c1的cout
      return 0;
}
运行结果为:
(8+14i)
(2+4i)
(6+10i)
```

```
Complex Complex::operator+(Complex &c2)  //定义运算符“+”重载函数
{ return Complex(real+c2.real,imag+c2.imag);}
ostream& operator << (ostream& output, Complex& c) //定义运算符“<<”重载函数
{  output<<"("<<c.real<<"+"<<c.imag<<"i)" <<endl;
   return output;
}
```

17

运算符重载

流运算符的重载-非常重要的例子

```
#include <iostream>
using namespace std;
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<<" +" <<c.imag<<" i)" ;
   return output;
}
istream& operator >> (istream& input, Complex& c)
{ cout<<"input real part and imaginary part of complex:";
  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;
}
```

貌似很复杂，其实都是套路

cout << obj <=> <<(cout, obj)

1、参数1：ostream &, istream &

2、参数2：输出、输入对象

3、返回类型： ostream &, istream &

4、实现逻辑：成员变量拼接到ostream

优化程序，虚部如果是负数，输出（4-10i） 而不是（4+-10i）

流运算符的重载-Demo

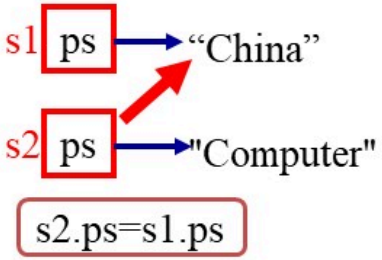
- class Clock{
  - int hour, min, sec;
  - friend ostream& operator << (ostream&, const Clock &);
    friend istream& operator >> (istream&, const Clock &);
- }

```
ostream& operator << (ostream& output, const Clock & c) {
    output <<c.hour<<" :" <<c.min<<" :" << c.sec;
    return output;
}
istream& operator >> (istream& input, const Clock & c)
{ cout<<"please input .....:";
  input>>c.hour>>c.min >> c.sec;
  return input;
}
```

19

赋值运算符重载

```
class  A{
    char *ps;
    public:       A( ){ ps=0;}
    A(char *s ){ ps =new char [strlen(s)+1];  strcpy(ps,s);}
    ~A( ){ if (ps)   delete [] ps;}
    void Show(void) {  cout<<ps<<endl;}
};
void  main(void ){
  A s1("China!"),s2("Computer!");
  s2=s1; //b.operator=(a);
  s1.Show();
  s2.Show();
}
```



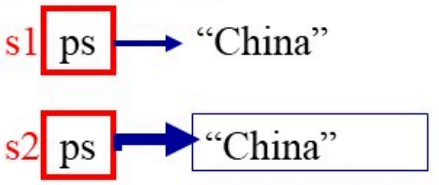
首先析构s2　接着析构s1出错

- C++会为每个类重载=运算符
- **注意区分=运算符与复制构造函数**
  - A a1, a2, a3= a1;
  - a2=a1; //赋值运算符
- 当对象的成员中使用了动态的数据类型时(用new开辟空间），必须要重载赋值运算符“=”，否则在程序的执行期间会出现运行错误。

赋值运算符重载

```
class A{
   char *ps;
   public:          A( ){ ps=0;}
   A(char *s ){ps =new char [strlen(s)+1];
     strcpy(ps,s);}
   ~A( ){ if (ps)   delete [] ps;}
   void Show(void) {  cout<<ps<<endl;}
   A&  operator=(A  &b);
};
void  main(void )
{ A s1("China!"),s2("Computer!");
  s1.Show();  s2.Show();
  s2=s1;
  s1.Show();  s2.Show();
}
```

```
A &  A::operator  = (  A  &b)//重载赋值运算符
{   if  ( ps )  delete  [ ] ps;
   if  ( b.ps)
   {     ps = new  char [ strlen(b.ps)+1];
         strcpy( ps, b.ps);
   }
   else          ps =0;
   return *this;
}
```



为什么返回A&，而不是void呢

s3=s2=s1; =》 s3.op=(s2.op=(s1))

s3.op=(void)错误，s3.op=(s1)

多选题 10分

- C++类的缺省函数包括哪些

A 构造函数

B 析构函数

C 复制构造函数

D 赋值运算符（重载）函数

22

类型转换函数重载

- 类型转换必须重载为类的成员函数，不能是友元函数
- 类型转换函数没有参数，被转换的是用户自定义对象。

A → double

A :: operator double ( )

{ 。 。 。 。 。 。 }

转换算法自己定义

```
class Complex{
        double Real,Image;
public:
    Complex(double real=0, double image=0)
        {Real=real;Image=image;}
     void Show(void)
      {。 。 。        }
     operator double();     //转换 Complex－> double
};
Complex::operator double ()
{       return Real*Real+Image*Image;}
void main(void)
{       Complex c(10,20);
        double f = c + 5; //c转换为double
        cout<<f<<endl;
}
```

类型转换函数重载

```
#include <iostream>
using namespace std;
class Complex{
   double Real, Image;
public:
   Complex(double r = 0, double i = 0) : Real(r), Image(i) {}
   operator double(); //?? Complex?> double
   friend Complex operator+(Complex, double);
};
Complex::operator double(){
   return Real * Real + Image * Image;
}
Complex operator+(Complex c1, double d){
   return Complex(c1.Real + d, c1.Image);
}
```

```
Complex c(1.0, 2.0);
double d1 = 3.0 + c; //类型转换
//引入类型转换可能造成混乱，二义性
double d2 = c + 3.0;//先重载+，后类型转换，赋值
double d3 = c - 3.0; //没重载-，所以类型转换

cout << d1 << endl; // 8
cout << d2 << endl; // 20
cout << d3 << endl; // 2
```

24

下一章 多态