第4讲　　运算符重载

回顾函数的重载：是指完成不同功能的函数可以具有相同的函数名。

C++的编译器是根据函数的实参来确定应该调用哪一个函数的。

```
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

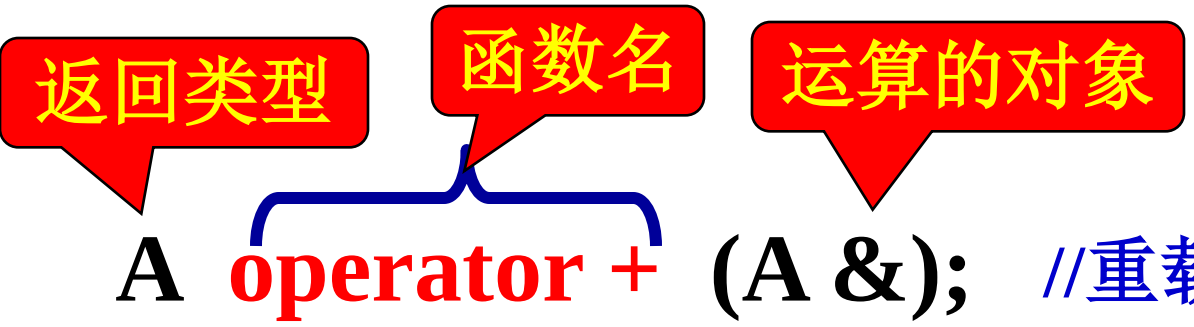
```
class A
{  float x,y;
public:
    A(float a=0, float b=0){  x=a; y=b; }
}
void main(void)
{  A   t1(2,3),  t2(3,4),  t3;
     c=t1+t2;
}
```

两对象不能使用＋，必须重新定义＋

运算符重载：

就是赋予已有的运算符多重含义。

重载函数为类的成员函数：




```
A operator +  (A &);   //重载了类A的“+”运算符
```

operator与其后的运算符一起构成函数名。

没有重载运算符的例子

```
class A
{       int i;
public:A(int a=0)      { i=a;  }
       void Show(void){      cout<<"i="<<i<<endl;       }
       void AddA(A &a,      A &b) //利用函数进行类之间的运算
       {       i=a.i+b.i;      }
};
void main(void)
{       A a1(10),a2(20),a3;
       a1.Show ();
       a2.Show ();
//     a3=a1+a2;                    //错，不可直接运算
       a3.AddA(a1,a2);              //对，调用专门的功能函数
       a3.Show ();
}
```

利用函数完成了加法运算

调用函数

有重载运算符的例子

```
class A
{       int i;
public:A(int a=0){ i=a;         }
       void Show(void){     cout<<"i="<<i<<endl;        }
       void AddA(A &a,   A &b)    //利用函数进行类之间的运算
       {       i=a.i+b.i;       }
       A operator + (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;
       a1.Show ();
       a2.Show ();
       a3=a1+a2;     //对，重载加法，可以直接进行类的运算
       a3.AddA(a1,a2);                  //调用专门的功能函数
       a3.Show ();
}
```

相当于
a3=a1.operator+(a2)

重载运算符与一般函数的比较：

相同：1）均为类的成员函数；2）实现同一功能

返回值　函数名　形参列表

```
void  AddA(A &a,   A &b)
{       i=a.i+b.i;   }
```

函数调用：

```
a3.AddA(a1,a2);
```

由对象a3调用

返回值　函数名

```
A operator +(A &a)
{   A   t;
    t.i=i+a.i;
    return t;
}
```

形参

函数调用：

```
a3=a1+a2;
a3=a1.operator+(a2);
```

由对象a1调用

在C++中,允许重载的运算符,不允许重载的运算符见书上**P125**页。

只能对C++中已定义了的运算符进行重载，而且，当重载一个运算符时，该运算符的优先级和结合律是不能改变的。

```
class room{
        float Length;
        float Wide;
public:  room(float a=0.0,float b=0.0){  Length=a;   Wide=b; }
  void Show(void){cout<<"Length="<<Length<<'\t'<<"Wide="<<Wide<<endl;}
  void ShowArea(void){  cout<<"Area="<<Length*Wide<<endl;   }
  room operator+(room &);//重载运算符+，函数原型
};
room room::operator + (room &r)  //重载运算符，函数定义
{  room rr;  //该函数要返回的room类对象rr
   rr.Length =Length+r.Length;
   rr.Wide =Wide+r.Wide ;
   return rr;
}
void main(void)
{  room r1(3,2),r2(1,4), r3,r4;
   r1.Show ();    r2.Show ();
   r3=r1+r2;                r3.Show ();
   r4=r1+r2+r3;             r4.Show ();}
```

r4=r1+r2+r3;

先(r1+r2); 再(r1+r2)+r3;

r4=r1+(r2+r3);

先(r2+r3); 再r1+(r2+r3);

运算符的优先级和结合律是不能改变的

```
class A
{       int i;
public:A(int a=0){ i=a;       }
       void Show(void){     cout<<"i="<<i<<endl;       }
       A operator +(A &a)  //重载运算符+
       {       A   t;   t.i=i+a.i;       return t;       }
       void operator+= (A &a)
       {       i=i+a.i;       }
};
void main(void)
{       A a1(10),a2(20),a3;
       a1.Show ();
       a2.Show ();
       a3=a1+a2;
       a1+=a2;
       a3.Show ();
}
```

由左操作符调用右操作符，没有返回值，故函数类型为void。

相当于a3=a1.operator+(a2)

相当于a1.operator+=(a2)

单目运算符的重载

前置和后置自增（自减）运算后对象a的值相同，但重载运算符函数的返回值不相同。

```
A   a;      A   a, b;
++a;        b=++a;      A operator ++(  ){ .... }
a++;        b=a++;      A operator ++(int){ .... }
```

C++约定：增加一个int类型的形参，表示：后置自增运算符重载函数

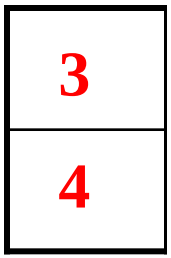
```
class A
{ float x, y;
public:
   A(float a=0, float b=0){  x=a;  y=b;  }
   A operator ++( ){A  t;  t.x=++ x; t.y=++y;  return t;}
  A operator ++(int) {  A  t;  t.x=x++;  t.y=y++;  return t;}
};
void main(void)
{  A   a(2,3), b;
    b=++a;
    b=a++;
}
```

返回值

函数名

```
A operator ++( )
{  A  t;
   t.x=++ x;
   t.y=++y;
   return t;
}
```

a




```
b=++a;
```

t作为函数值返回赋给b

或者：

```
A operator ++( )
{   ++ x;
    ++y;
    return *this;
}
```

将对象本身作为函数值返回赋给b

返回值

函数名

```
A operator ++(int)
{  A  t;
   t.x=x++;
   t.y=y++;
   return t;
}
```

a

3
4

```
   b=a++;
```

t作为函数值返回赋给b

运算符重载为友元函数

运算符重载为成员函数时，是由一个操作数调用另一个操作数。

A a ,b , c;

c=a+b; 实际上是**c=a.operator+(b);**

c=++a; 实际上是**c=a.operator++( );**

c+=a; 实际上是**c.operator+=(a);**

重载+=

即函数的实参只有一个或没有。

友元函数是在类外的普通函数，与一般函数的区别是可以调用类中的私有或保护数据。

将运算符的重载函数定义为友元函数，参与运算的对象全部成为函数参数。

A a ,b , c;

c=a+b; 实际上是 **c=operator+(a, b);**

c=++a; 实际上是 **c=operator++(a);**

c+=a; 实际上是 **operator+=(c, a);**

```
class A
{       int i;
public:
        A(int a=0)      { i=a;  }
        void Show(void)     {       cout<<"i="<<i<<endl;        }
 friend A operator +(A &,A &);//友元函数，两个参数，为引用
};
A operator +(A &a , A &b) //类体外定义作为友元函数的重载函数
        {A t;   t.i=a.i+b.i;     return t; }
void main(void)
{       A a1(10),a2(20),a3;
        a1.Show ();
        a2.Show ();
        a3=a1+a2;     //重新解释了加法，可以直接进行类的运算
        a3.Show ();
}
```

相当于
a3=operator+(a1,a2)

```
class A
{       int i;
public:
        A(int a=0)      { i=a;  }
        void Show(void)     {       cout<<"i="<<i<<endl;        }
        friend A operator++(A &a){ a.i++;  retrurn a;}// 类体内定义
        friend A operator++(A &a, int )//类体内定义，int表示后增
        {  A  t;            t.i=a.i;   a.i++;       return t;}
};
void main(void)
{       A a1(10),a2,a3;
        a2=++a1;
        a3=a1++;
        a2.Show();      a3.Show ();
}
```

相当于a2=operator++(a1)

相当于a3=operator++(a1,int)

对双目运算符，重载为成员函数时，仅一个参数，另一个被隐含；重载为友元函数时，有两个参数，没有隐含参数。

一般来说，单目运算符最好被重载为成员函数；对双目运算符最好被重载友元函数。

转换函数

转换函数就是在类中定义一个成员函数，其作用是将类转换为某种数据类型。

利用转换函数将类A的对象a转换成某种数据类型

```
class  A
{    float x, y;
  public:
    A(float  a, float b){  x=a; y=b;  }
};
void main(void)
{  A   a(2,3);
   cout<<a<<endl;
}
```

错误！类的对象不能直接输出

A ⟶ float

A → float

1. 转换函数必须是类的成员函数。

格式为：

类名　关键字　欲转换类型

ClassName :: operator <type>( )

{.....}　具体的转换算法

2. 转换函数的调用是隐含的，没有参数。

A :: operator float ( )

{ return x+y; }　转换算法自己定义

```
class Complex{
        float Real,Image;
public:
    Complex(float real=0,float image=0)
        {       Real=real;     Image=image;         }
     void Show(void)
      {cout<<"Real="<<Real<<'\t'<<"Image="<<Image<<endl; }
 operator float();      //成员函数，定义类转换 Complex－>float
};
Complex::operator float ()
{       return Real*Real+Image*Image;}
void main(void)
{       Complex c(10,20);
        c.Show ();
        cout<<c<<endl;//可以直接输出c，因为已经进行类型转换

}
```

注意：转换函数只能是成员函数，不能是友元函数。转换函数的操作数是对象。

赋值运算符与赋值运算符重载 “＝”

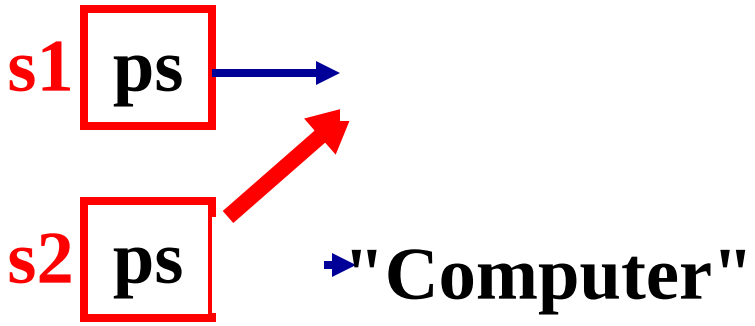
同类型的对象间可以相互赋值，等同于对象的各个成员的一一赋值。

```
A    a(2,3), b;

b＝a;
```

但当对象的成员中使用了动态的数据类型时(用new开辟空间），就不能直接相互赋值，否则在程序的执行期间会出现运行错误。

```
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;}//指针非空时
void Show(void) {  cout<<ps<<endl;}
};
void  main(void )
{ A s1("China!"),s2("Computer!");
  s1.Show();  s2.Show();
  s2=s1;  //相当于 s2.ps=s1.ps;
  s1.Show();  s2.Show();
}
```



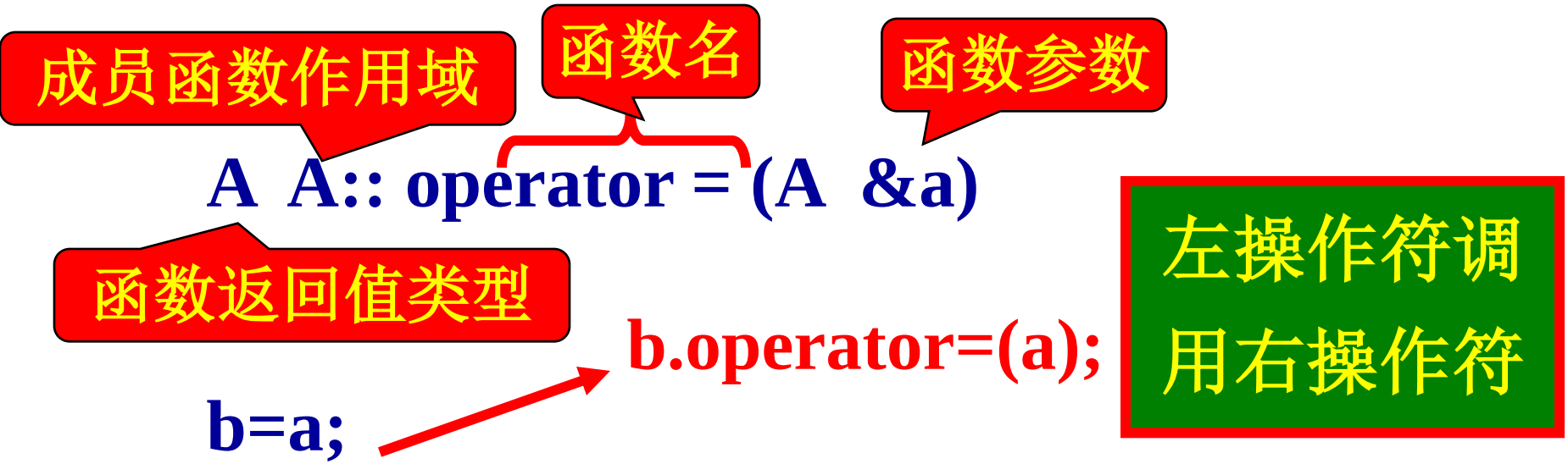
s2.ps=s1.ps

首先析构s2

接着析构s1出错

这时，利用编译系统的默认赋值无法正确运行程序，必须重载赋值运算符“＝”

赋值运算符必须重载为成员函数。



```
class Sample{
        int x;
public: Sample(int i=0){x=i;}
        void disp(void){  cout<<“x=“<<x<<endl;}
        void operator=(Sample &p)   {  x=p.x;  }
};
void main(void)
{   Sample  A(20),B;
    Sample   C(A);//使用缺省的拷贝构造函数
    B=A;         //使用赋值运算符重载
    B.disp();
    A.disp();
}
```

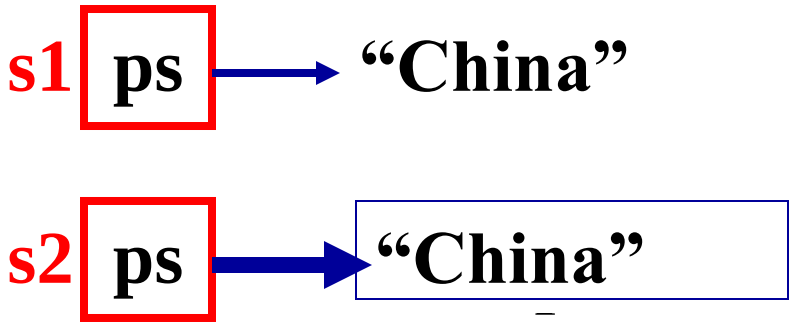
```
class  A{
        char *ps;
public:       A( ){ ps=0;}
A(char *s ){ps =new char [strlen(s)+1];  strcpy(ps,s);}
~A( ){ if (ps) delete ps;}
void Show(void) {  cout<<ps<<endl;}
A & operator=(A  &b);//重载函数声明，返回该类的引用
};
void  main(void )
{ A s1("China!"),s2("Computer!");
  s1.Show();  s2.Show();
  s2=s1;
  s1.Show();  s2.Show();
}
```

必须重新定义“=”

```
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;
}
```

s2=s1;

s2.operator=(s1);



返回同种类型的引用适合于连等式。

s3=s2=s1;

上述内容的完整程序

```
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;}
            char *GetS( ) {return ps;}
            A &   operator  = (  A  &b);//重载赋值运算符
};
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;
}
void  main(void )
{           A s1("China!"),s2("Computer!");
            s2=s1;
            cout <<"s1= "<< s1.GetS()<<'\t';
            cout <<"s2= "<< s2.GetS()<<'\n';
}
```

重新开辟内存

s2.ps重新开辟内存，存放“China”

作业

- P153第7题