面向对象程序设计
420420

运算符重载

- 1、运算符重载的概念
- 2、运算符函数

39.1 运算符重载的概念

- 运算符重载就是对现有的运算符重新进行定义，赋予其另一种功能，以适应不同的数据类型。

39.1 运算符重载的概念

- 所谓重载，就是重新赋予新的含义。例如函数重载，就是对一个已有的函数赋予新的功能。

- C++语言本身就重载了很多运算符，例如<<是位运算中的左移运算符，但是在输出操作中它与流对象cout配合，变成流插入运算符，>>运算符也是类似。

- C++允许程序员重载大部分运算符，使运算符符合所在的上下文环境。虽然重载运算符的任务也可以通过显式的函数调用来完成，但是使用运算符重载往往使程序更清晰。

- 本质上，运算符重载就是函数的重载。重载运算符是具有特殊名称的函数，形式如下：

```
返回类型    operator运算符号(形式参数列表)
{
        函数体
}
```

- operator后面接上需要重载的运算符，成为运算符函数。
- 运算符函数的函数名就是“operator运算符号”。

例：

```
class   complex     //复数类
{
        public:
                complex(double  r=0.0,double   i=0.0)  //构造函数
                {real=r;  imag=i;}
        private:
                double  real;  //实部
                double  imag;  //虚部
};
```

- 将 ‘+’用于复数类的加法运算，运算符函数原型可以是：

```
complex operator+(const complex &a,const complex &b);
```

- 调用运算符函数的形式如下：

```
operator 运算符号(实参列表);
```

- 例如：complex c1(1,2),c2(3,4),c3;
- c3=operator +(c1,c2);

- 以上虽然是规范的运算符函数调用方法，但程序员更喜欢写成：

c3=c1+c2;

- 重载运算符的规则
- （1）C++绝大部分的运算符可以重载，不能重载的运算符有：

  . .* :: ?: sizeof

- （2）不能改变运算符的优先级、结合型和运算对象数目。
- （3）运算符重载函数不能使用默认参数。
- （4）重载运算符必须具有一个类对象（或类对象的引用）的参数，不能全部是C++的内置数据类型。
- （5）一般若运算结果作为左值则返回类型为引用类型；若运算结果要作为右值，则返回对象。
- （6）重载运算符的功能应该与原来的功能一致。

运算符重载

- 3、运算符重载为类成员函数
- 4、运算符重载为友元函数

39.3 运算符重载为类的成员函数

- 将运算符重载为类的成员函数，一般形式为：

```
class   类名{     //类体
        ……
        返回类型   operator   运算符号(形式参数列表)
        {
                函数体
        }
        ……
};
```

- 或者：

```
class   类名{     //类体
        ……
        返回类型   operator   运算符号(形式参数列表);
        ……
};
返回类型   类名::operator   运算符号(形式参数列表)
{
        函数体
}
```

39.3 运算符重载为类的成员函数

- 当运算符重载为成员函数时，运算符函数的形式参数的个数比运算符规定的运算对象个数要少一个。

- 原因是类的非静态成员函数都有一个隐含的this指针，运算符函数可以用this指针隐式地访问类对象的成员，因此这个对象自身的数据可以直接访问，不需要放到形参列表中进行传递，少了的运算对象就是该对象本身。

39.3 运算符重载为类的成员函数

- 1. 双目运算符重载为类的成员函数，形式如下：

```
返回类型  类名::operator  op(const  所属类型  &obj2)
{
        ……              //this指针对应obj1运算对象
}
```

- 经过重载后，表达式 “obj1 op obj2”
- 相当于obj1.operator op(obj2)

39.3 运算符重载为类的成员函数

【例39.1】设计一个复数类，实现该类对象的加法和减法运算。

```
1  #include <iostream>
2  using namespace std;
3  class complex    //复数类
4  {
5  public:
6          complex(double r=0.0,double i=0.0){real=r; imag=i;}//构造函数
7          complex operator +(const complex &c2); //重载+运算符
8          complex operator -(const complex &c2); //重载-运算符
9          void display()
10         {
11                 cout<<real<<"+"<<imag<<"i"<<endl;
12         }
13 private:
14         double real; //私有数据成员，实部
15         double imag; //私有数据成员，虚部
16 };
```

39.3 运算符重载为类的成员函数

```
17  complex  complex::operator  +(const complex  &c2)
18  {       complex  c;
20          c.real=c2.real+real;
21          c.imag=c2.imag+imag;
22          return  c;
23  }
24  complex  complex::operator  -(const complex  &c2)
25  {       complex  c;
27          c.real=real-c2.real;            //顺序不能颠倒
28          c.imag=imag-c2.imag;        // this指针所隐藏的对象是指的第一个操作数
29          return  c;
30  }
31  int  main()
32  {       complex  c1(1,2),c2(3,4),c3;
34          c3=c2+c1;     c3.display();     //输出4+6i
36          c3=c2-c1;     c3.display();     //输出2+2i
38          return  0;
39  }
```

- 2. 前置单目运算符重载为类的成员函数，形式如下：

```
返回类型   类名::operator  op()
{
        ……              //this指针对应obj运算对象
}
```

- 经过重载后，表达式 “op obj”
- 相当于obj.operator op()

39.3 运算符重载为类的成员函数

【例39.2】实现复数类的前置自增运算。

```
1  #include <iostream>
2  using namespace std;
3  class complex    //复数类
4  {
5  public:
6          complex(double r=0.0,double i=0.0){real=r; imag=i;}//构造函数
7          complex operator ++(); //重载前置++
8          void display()
9          {
10                 cout<<real<<"+"<<imag<<"i"<<endl;
11         }
12 private:
13         double real; //实部
14         double imag; //虚部
15 };
```

39.3 运算符重载为类的成员函数

```
16  complex  complex::operator  ++()
17  {
18          complex  a;
19          real++;
20          imag++;
21          a.real=real;
22          a.imag=imag;
23          return  a;
24  }
25  int  main()
26  {
27          complex  c1(1,2),c2;
28          c2=++c1;
29          c1.display();     //输出2+3i
30          c2.display();     //输出2+3i
31      return  0;
32  }
```

- 3. 后置单目运算符重载为类的成员函数，形式如下：

```
返回类型   类名::operator  op(int)
{
        ……              //this指针对应obj运算对象
}
```

- 经过重载后，表达式 “obj op”
- 相当于obj.operator op(0)

39.3 运算符重载为类的成员函数

【例39.3】实现复数类的后置自增运算。

```
1  #include <iostream>
2  using namespace std;
3  class complex    //复数类
4  {
5  public:
6          complex(double r=0.0,double i=0.0) {real=r; imag=i;} //构造函数
7          complex operator ++(int); //重载后置++
8          void display()
9          {
10                 cout<<real<<"+"<<imag<<"i"<<endl;
11         }
12 private:
13         double real; //实部
14         double imag; //虚部
15 };
```

39.3 运算符重载为类的成员函数

```
16  complex complex::operator ++(int)
17  {
18          complex a;
19          a.real=real;
20          a.imag=imag;
21          real++;
22          imag++;
23          return a;
24  }
25  int main()
26  {
27          complex c1(1,2),c2;
28          c2=c1++;
29          c1.display();    //输出2+3i，c1自增后的值
30          c2.display();    //输出1+2i，c1自增前的值
31          return 0;
32  }
```

39.4 运算符重载为友元函数

- 当运算符重载为友元函数时，运算符函数的形式参数的个数和运算符规定的运算对象个数一致。
- 形式如下：

```
class  类名{     //类体
        ……
        //友元声明
        friend  返回类型  operator  运算符号(形式参数列表);
};
返回类型  operator  运算符号(形式参数列表)  //函数定义
{
        函数体
}
```

- 1. 双目运算符重载为类的友元函数，形式如下：

```
返回类型 operator op(const 所属类型 &obj1,const 所属类型 &obj1)
{
        ……    //  obj1和obj2分别对应两个运算对象
}
```

- 经过重载后，表达式 “obj1 op obj2”
- 相当于operator op(obj1,obj2)

39.4 运算符重载为友元函数

【例39.4】将加减运算符重载为复数类的友元函数。

```
#include <iostream>
using namespace std;
class complex    //复数类
{
public:
  complex(double r=0.0,double i=0.0) {real=r; imag=i;}//构造函数
  friend complex operator +(const complex &c1,const complex &c2); //重载+运算符
  friend complex operator -(const complex &c1,const complex &c2); //重载-运算符
  void display()
            {   cout<<real<<"+"<<imag<<"i"<<endl; }
private:
      double real; //实部
      double imag; //虚部
};
```

39.4 运算符重载为友元函数

```
15 complex operator +(const complex &c1,const complex &c2)
16 {     complex c3;                    //complex对象c3
17       c3.real=c1.real+c2.real;
18       c3.imag=c1.imag+c2.imag;
19       return c3;
20 }
21 complex operator -(const complex &c1,const complex &c2)
22 {     complex c3;                    //complex对象c3
23       c3.real=c1.real-c2.real;
24       c3.imag=c1.imag-c2.imag;
25       return c3;
26 }
27 int main()
28 {     complex c1(1,2),c2(3,4),c3;
29       c3=c2+c1;    c3.display();    //输出4+6i
30       c3=c2-c1;    c3.display();    //输出2+2i
31       return 0;
32 }
```

- 2. 前置单目运算符重载为类的友元函数，形式如下：

```
返回类型  operator  op(const  所属类型  &obj)
{
        ……    //  obj对应运算对象
}
```

- 经过重载后，表达式 “op obj2”
- 相当于operator op(obj)

- 3. 后置单目运算符重载为类的友元函数，形式如下：

```
返回类型  operator  op(const  所属类型  &obj, int)
{
        ……    //  obj对应运算对象
}
```

- 经过重载后，表达式 “obj op”
- 相当于operator op(obj,0)

39.4 运算符重载为友元函数

【例39.5】将前置自增和后置自增运算符重载为复数类的友元函数。

```
#include <iostream>
using namespace std;
class complex    //复数类
{
public:
        complex(double r=0.0,double i=0.0){real=r; imag=i;}//构造函数
        friend complex operator ++(const complex &c1); //重载前置++
        friend complex operator ++(const complex &c1,int); //重载后置++
        void display()
        {
                cout<<real<<"+"<<imag<<"i"<<endl;
        }
private:
        double real; //实部
        double imag; //虚部
};
```

39.4 运算符重载为友元函数

```
17  complex  operator  ++(const complex  &c1)
18  {
19          complex  c;
20          c1.real++;
21          c1.imag++;
22          c.real=c1.real;
23          c.imag=c1.imag;
24          return  c;
25  }
26  complex  operator  ++(const complex  &c1,int)
27  {
28          complex  c;
29          c.real=c1.real;
30          c.imag=c1.imag;
31          c1.real++;
32          c1.imag++;
33          return  c;
34  }
```

39.4 运算符重载为友元函数

```
35 int main()
36 {
37         complex c1(1,2),c2,c3;
38         c2=++c1;          //前置自增
39         c1.display();     //输出2+3i
40         c2.display();     //输出2+3i
41         c3=c1++;          //后置自增
42         c1.display();     //输出3+4i
43         c3.display();     //输出2+3i
44         return 0;
45 }
```