Ch-10

运算符重载

主要内容

- 什么是运算符重载
- 运算符重载的方法
- 重载运算符的规则
- 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

10.1 什么是运算符重载

□ 定义

运算符重载是对已有的运算符赋予新的含义，使之可以实现对指定类型的指定运算。

运算符重载的实质是函数重载：通过创建运算符函数，重新定义重载运算符需要实现的运算功能。

□ 实现机制

首先把指定的运算表达式转化为对运算符函数的调用，运算对象转化为运算符函数的实参，然后编译系统根据实参的类型，遵循函数重载的选择原则确定需要调用的运算符函数。

10.2 运算符重载的方法

□ 运算符重载函数的定义

函数类型 operator 运算符(形参表)

{ 重载处理 }

其中，operator是关键字，“operator 运算符”是函数名。

例如：Complex operator + (Complex &c2);

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

10.3 运算符重载的规则：

(1) 5种不允许重载的运算符：成员运算符“.”、成员指针运算符“.*”、作用域运算符“::”、sizeof运算符和条件运算符“?:”；

(2) 重载运算符必须是C++语言中已有的且允许重载的那些运算符，不能创造新的运算符；

(3) 运算符重载实质上是函数重载，因此编译系统对运算符重载的选择，遵循函数重载的选择原则，但需要注意运算符重载函数不能带默认值参数；

(4) 重载之后的运算符不能改变原有的优先级和结合性，也不能改变操作数的个数及语法结构；

(5) 运算符重载不会改变该运算符用于内置类型的含义，但应用运算符重载时要求至少有一个操作数是用户自定义类型，如类对象或类对象的引用；

(6) 运算符重载是针对新类型数据的实际需要对原有运算符进行的适当的改造，重载的功能应当与原有功能相类似，避免没有目的地使用重载运算符；

(7) C++默认提供 = 和 & 的运算符重载；

(8) C++规定赋值运算符、下标运算符[]、函数调用运算符()必须定义为类的成员函数；而输出流插入<<、输入流提取>>、类型转换运算符不能定义为类的成员函数。

10.4 运算符重载的使用方式

运算符重载的使用方式有两种：作为类成员函数或友元函数。

具体使用方式如下表所示：

运算符	使用方式
所有一元运算符	建议是成员函数
= ( ) [ ] ->	必须是成员函数
+= -= /= *= ^= &= != %= >>= <<=	建议是成员函数
所有其它二元运算符, 例如: –, +, *, /	建议是友元函数
<< >>	必须是友元函数

```
4   class Complex
5   {
6       public:
7           Complex():real(0), imag(0) {}
8           Complex(double r, double i):real(r), imag(i) {}
9           /*
10          *运算符重载为成员函数的情况下，参数表中隐含了第一个参数：this指针，
11          *通过this指针能够直接访问本类中私有成员
12          */
13          Complex operator +(const Complex &c);
14      private:
15          double real, imag;
16  };
17
18  Complex Complex::operator +(const Complex &c)
19  {
20      return Complex(real + c.real, imag + c.imag);
21  }
```

```
4   class Complex
5   {
6       public:
7           Complex():real(0), imag(0) {}
8           Complex(double r, double i):real(r), imag(i) {}
9           /*
10          *运算符重载为友元函数的情况下，不能省略参数
11          */
12          friend Complex operator +(const Complex &c1, const Complex &c2);
13      private:
14          double real, imag;
15  };
16
17  Complex operator +(const Complex &c1, const Complex &c2)
18  {
19      return Complex(c1.real + c2.real, c1.imag + c2.imag);
20  }
```

例10.1

```
30  int main()
31  {
32      Complex c1(2.0, 3.0), c2(4.0, -2.0), c3;
33      /*
34      *c3 = c1 + c2是运算符重载的隐式调用形式，其中
35      *运算符"+"左侧的操作数对应第一个函数参数，
36      *运算符"+"右侧的操作数对应第二个函数参数。
37      *（1）在运算符重载是成员函数的情况下，
38      *其等价的显示调用形式为c3 = c1.operator+(c2);
39      *（2）在运算符重载是友元函数的情况下，
40      *其等价的显示调用形式为c3 = operator +(c1, c2);
41      */
42      c3 = c1 + c2;
43      cout << "\nc1+c2=";
44      print(c3);
45      cout << endl;
46      return 0;
47  }
```

◆ 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（即第一个操作数）必须是Complex类对象，程序中可以写成：

```
c3 = c2 + n;  //不能写成c3 = n + c2;
```

- 如果要求运算符左侧操作数不是对象，则应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对象，但要求实参与形参一一对应：

```
c3 = n + c2;  //正确
c3 = c2 +n;   //错误
```

因此，在重载加法运算符的时候，如果两个操作数分别是类对象和内置类型，则一般不满足交换律。如果为了满足交换律，必须定义两个运算符重载函数。

10.5 双目运算符重载

例10.2 利用关系运算符重载实现字符串的比较

□ main函数的定义

```
5   int main(int argc, char** argv)
6   {
7       String str1("Hello"), str2("Book"), str3("Computer"), str4("Computer");
8       compare(str1, str2);
9       compare(str2, str3);
10      compare(str3, str4);
11      return 0;
12  }
```

□ compare函数的定义

```
40  void compare(String &str1, String &str2)
41  {
42      if(str1>str2) { //operator>(str1, str2)
43          str1.display();
44          cout << ">";
45          str2.display();
46      } else if(str1<str2) {  //operator<(str1, str2)
47          str1.display();
48          cout << "<";
49          str2.display();
50      } else if(str1==str2) { //operator==(str1, str2)
51          str1.display();
52          cout << "=";
53          str2.display();
54      }
55      cout << endl;
56  }
```

□类的声明及运算符重载函数的定义

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

void compare(String &, String &);
```

```
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;
}
```

10.6 单目运算符重载

单目运算符只需要一个操作数，因此重载函数最多只有一个参数。如果将运算符重载函数定义为成员函数则可以没有参数。

下面以自增运算符++为例，学习单目运算符的重载函数的编写方法。

例10.3 有一个Time类，数据成员有时、分、秒。要求模拟秒表，每次走一秒，满60秒进位，秒又从零开始计数。满60分进位，分又从零开始计数。输出时、分和秒的值。

□ main函数的定义

```
5   int main()
6   {
7       Time time1(21,34,59), time2;
8       cout<<" time1 : ";
9       time1.display();
10      ++time1;
11      cout<<"++time1: ";
12      time1.display();
13      time2 = time1++;
14      cout<<"time1++: ";
15      time1.display();
16      cout<<" time2 : ";
17      time2.display();
18      return 0;
19  }
```

```
 time1 : 21:34:59
++time1: 21:35:0
time1++: 21:35:1
 time2 : 21:35:0
```

□ Time类的声明

```
4   class Time
5   {
6       public:
7           Time() {
8               hour=0;
9               minute=0;
10              sec=0;
11          }
12          Time(int h, int m, int s): hour(h), minute(m), sec(s) {}
13          Time operator++();  //前置++运算符重载函数
14          Time operator++(int); //后置++运算符重载函数
15          void display();
16      private:
17          int hour ;
18          int minute;
19          int sec;
20  };
```

□ 运算符重载函数的定义

```
5   Time Time::operator++() //前置++运算符重载函数
6   {
7       sec++;
8       if(sec>=60) {
9           sec=sec-60;
10          minute++;
11          if(minute>=60) {
12              minute=minute-60;
13              hour++;
14              hour=hour%24;
15          }
16      }
17      return *this;
18  }
19
20  Time Time::operator++(int)  //后置++运算符重载函数
21  {
22      Time temp(*this);  //保存修改前的对象做返回值
23      ++(*this);
24      return temp;
25  }
```

- **在++或--的运算符重载函数中：如果没有形参，则是前置重载函数；如果有一个int型形参，则是后置重载函数。（这个int型形参只是为了区别前置和后置，而没有其他作用，因此在括号里写int即可）**

10.7 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，并且在类库中对左移运算符和右移运算符进行了重载，作为流插入运算符“<<”和流提取运算符“>>”。但它们只能用于内置类型的数据，为了实现自定义类型的输入和输出，就需要对它们进行重载。

□ 重载函数的原型：

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

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

这两个函数只能定义为友元函数，而不能定义为成员函数，因为函数有两个形参，并且第一个形参不是自定义类型。

10.7.1 重载流插入运算符“<<”

例10.4 在例10.1的基础上，重载流插入运算符“<<”输出复数

- **在类中声明<<重载函数是友元函数**

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

- **在类外定义友元函数：**

```
ostream& operator << (ostream& output, Complex& c) {
        output<<"("<<c.real;
        if(c.imag >= 0) output<<"+";
        output<<c.imag<<"i)"<<endl;
        return output; //将输出流现状返回，以便连续输出
}
```

- **main函数的定义**

```
int  main() {
        Complex c1(2,4),c2(6,10),c3;
        c3=c1+c2;
        cout<<c3;  //编译系统解释为：operator<<(cout, c3);，调用重载函数
        return 0;
}
```

10.7.2 重载流提取运算符“>>”

例10.5 在例10.4中增加流提取运算符>>重载函数，用cin>>输入复数，用cout<<输出复数。

- **在类中声明>>重载函数是友元函数：**

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

- **在类外定义函数：**

```
istream& operator >> (istream& input, Complex& c) {
        cout << "请输入复数的实部和虚部:";
        input >>c.real >>c.imag;
        return input;
}
```

- **main函数的定义**

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

◆从本章例子中可以注意到，在运算符重载中使用引用参数的重要性，用引用形参在调用函数时，通过传递地址方式让形参成为实参的别名，不用生成临时变量，减少了时间和空间的开销。此外，如重载函数的返回值是对象引用时，返回的是对象，它可以出现在赋值号的左侧而成为左值，可以被赋值或参与其他操作（如保留 cout流的当前值以便能连续使用<<输出）。

10.8 不同类型数据间的转换

10.9.1 内置基本类型数据间的转换

- 隐式类型转换：编译系统自动转换
- 强制类型转换：手动转换，int(x) 或 (int)x。

10.9.2 用户自定义的类类型与其他数据类型间的转换

10.9.2.1 用类型转换构造函数进行数据类型转换

总结已学习的三种构造函数：

- 默认构造函数：Complex(); //没有参数
- 用于初始化的构造函数：Complex(double r, double i); //一般有两个以上形参
- 用于复制对象的拷贝构造函数：Complex(Complex &c); //形参是本类对象的引用

□ 类型转换构造函数

- 类型转换构造函数是将其他类型转换成类类型的构造函数，它只能有一个参数。

例如：

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

该构造函数将double型数据r转换成Complex类的对象，在这个对象中r作为实部，虚部为0。

- 通常将只有一个参数的构造函数作为类型转换构造函数，而不做其他用途。
- 定义了类型转换构造函数后，可以采用以下形式创建对象：

```
Complex c1(3.6);  //调用类型转换构造函数建立对象（即类型转换）

Complex (3.6);  //建立了一个无名对象，无法直接引用
```

□ 类型转换函数

类型转换函数是将类类型转换为其他类型的函数。

- 类型转换函数的一般形式

operator 类型名() {实现转换的语句}

说明：（1）类型转换函数不能指定函数类型，它的返回值类型是由operator后面的类型名来确定的；（2）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。

当需要使用类型转换构造函数或类型转换函数时，编译系统会自动调用这些函数，建立一个无名的临时对象（或临时变量）。

例10.5（部分代码）

```
4   class Complex
5   {
6       public:
7           Complex():real(0), imag(0) {}   //默认构造函数，无形参
8           Complex(double r):real(r), imag(0) {}   //类型转换构造函数，一个形参
9           Complex(double r, double i):real(r), imag(i) {} //普通构造函数，两个形参
10          friend Complex operator +(const Complex &, const Complex &);//友元运算符重载函数
11          void display();
12      private:
13          double real, imag;
14  };
15
16  Complex operator+(const Complex &c1, const Complex &c2)
17  {
18      return Complex(c1.real + c2.real, c1.imag + c2.imag);
19  }
```

```
26  int main()
27  {
28      Complex c1(2.0, 3.0), c2(4.0, -7.0), c3, c4;
29      c3 = c1 + c2;
30      c4 = c2 + 6.7;
31      c3.display();
32      c4.display();
33      return 0;
34  }
```

CH10 End