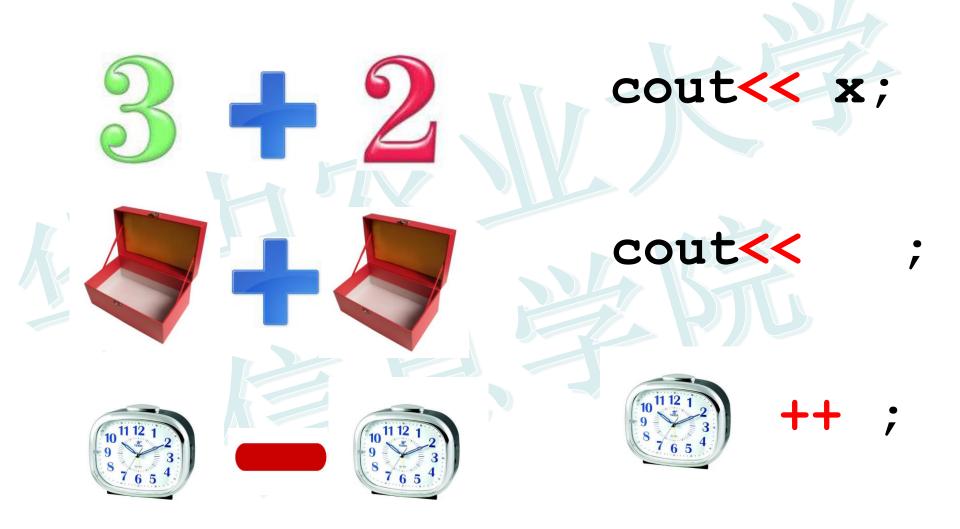
C/C++程序设计案例实战 一加减乘除之多变

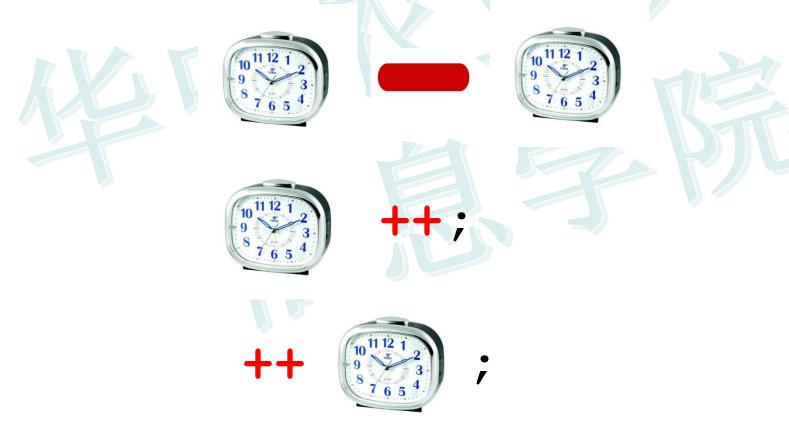
华中农业大学信息学院 章 英

问题引入



运算符重载

• 对已有的运算符赋予多重含义,使同一个运算符作用于不同类型的数据时导致不同的行为。



多态性

- 多态是指操作接口具有表现多种形态的能力,即 能根据操作环境的不同采用不同的处理方式。
- 多态性是面向对象系统的主要特性之一,在这样的系统中,一组具有相同基本语义的方法能在同一接口下为不同的对象服务。

• 例如:

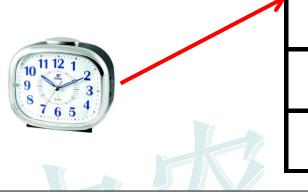
一同样的消息——相加,被不同类型的对象——变量接收后,不同类型的变量采用不同的方式进行加法运算。

多态分类

• 多态的类型

- 重载多态: 函数重载、运算符重载
- 强制多态: 浮点数和整数相加, 进行强制类型转换
- 包含多态:不同类中的同名成员函数,虚函数实现
- 参数多态: 类模板实例化时操作对象的类型不同
- ・根据多态性作用的时机可以分为
 - 编译时的多态:编译过程中确定同名操作的具体对象
 - 运行时的多态:程序运行过程中确定操作的具体对象

案例——代码实现



hour minute second



```
11.1 运算符重载.cpp - Code::Blocks 17.12
le <u>E</u>dit <u>V</u>iew Sea<u>r</u>ch <u>P</u>roject <u>B</u>uild <u>D</u>ebug Fortra<u>n</u> <u>w</u>xSmith <u>T</u>ools T<u>o</u>ols+ Plugins Do<u>x</u>yBlocks <u>S</u>ettings <u>H</u>elp
                   🔤 🖫 📔 🕒 🔡 🎒 🕲 🦫 🐰 🐚 🧥 🔍 🔍
Start here × 11.1 运算符重载.cpp ×
            #include <iostream>
            using namespace std;
            class Clock { //时钟类声明定义
            public: //外部接口
                 Clock(int hour = 0, int minute = 0, int second = 0);
                 void showTime() const;
                 Clock operator-(const Clock &c2) const;
                 Clock& operator ++ (); //前置单目运算符重载
                 Clock operator ++ (int); //后置单目运算符重载
     10
            private: //私有数据成员
     11
     12
                 int hour, minute, second;
     13
```

时钟类——重载减法运算符

声明形式:

```
函数类型 类名::operator 运算符 (形参) { .... }
```

```
Clock Clock::operator-(const Clock &c2)
const
{
    return Clock(hour-c2.hour ,
    minute-c2.minute, second-c2.second);
}
```

两种途径

- 重载为类的成员函数时
 - 参数个数=原操作数个数-1 (后置++、--除外)
- 重载为非成员函数时
 - 参数个数=原操作数个数,且至少应该有一个自定义 类型的形参。

```
Clock myClock1(23, 59, 59);
Clock myClock2(23, 50, 11);
例如: myClock1 - myClock2
myClock1.operator-(myClock2)
```

对象1

函数名

参数 (对象2)

时钟类——重载减法运算符



函数名

两个参数(对象1和对象2)

```
Clock operator-(const Clock &c1, const
Clock &c2)
{
    return Clock(c1.hour-c2.hour
        c1.minute-c2.minute,
        c1.second-c2.second);
}
```

hour-c2.hour

声明为友元函数

- } ;



int hour, minute, second;

```
Clock& Clock::operator ++ ();
Clock Clock::operator ++ (int);
```

- 前置单目运算符,重载函数没有形参,
- · 后置单目运算符, 重载函数需要有一个整型形参。

```
Clock & Clock::operator ++ () { //前置单目运算符重载函数
34
35
         second++;
36
         if (second >= 60) {
                                        ++myClock1
37
            second -= 60; minute++;
38
            if (minute >= 60) {
39
                minute -= 60;
40
                hour = (hour + 1) % 24;
41
42
43
         return *this;
44
     Clock Clock::operator ++ (int) { //后置单目运算符重载
45
46
         //注意形参表中的整型参数
47
        Clock old = *this;
        ++(*this); //週用前置<u>\</u>++"运算符
48
49
        return old:
                                       myClock1++
50
```

• 重载为非成员函数

```
方法2
```

```
Clock operator++(Clock &);
Clock operator++(Clock &, int);
```



```
54
      Clock operator++ (Clock &c)
55
          c.second++;
56
          if (c.second \geq 60) {
57
              c.second -= 60; c.minute++;
58
              if (c.minute \geq 60) {
59
                   c.minute -= 60;
60
                   c.hour = (c.hour + 1) % 24;
61
62
63
          return c;
64
65
      Clock operator++(Clock &c, int)
66
         Clock old = c;
          ++(c); //週用前置<u>"</u>++"运算符
67
68
          return old;
69
```

声明为友元函数

```
//时钟类声明定义
     class Clock {
 5
     public: //处部接口
         Clock(int hour = 0, int minute = 0, int second = 0);
         void showTime() const;
 8
         //Clock operator-(const Clock &c2) const;
 9
         //Clock& operator ++ (); //前置单目运算符重载
10
         //Clock operator ++ (int); //后置单目运算符重载
         friend Clock operator-(const Clock &, const Clock &);
11
12
         friend Clock operator++(Clock &);
13
         friend Clock operator++(Clock &, int);
     private: //私有数据成员
14
15
         int hour, minute, second;
16
```

案例代码——运行结果

23:59:59 — 23:50:11 23:59:59 + + + + 00:00:00

```
0:9:48
First time output: 23:59:59
Show myClock++: 23:59:59
Show ++myClock: 0:0:1
```

```
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
                  | 🗟 | | 🖰 🕒 🔒 🎒 | 🐍 🦫 | 🐰 🖿 🖺 | 🔍 🙉
Start here × 11.1 运算符重载.cpp ×
      77
             int main() {
                  Clock myClock1(23, 59, 59);
      78
      79
                  Clock myClock2 (23, 50, 11);
      80
                   (myClock1-myClock2) .showTime();
                  cout << "First time output: ";</pre>
      81
      82
                  myClock1.showTime();
      83
                  cout << "Show myClock++:</pre>
                                                         ";
      84
                   (myClock1++) .showTime();
                  cout << "Show ++myClock:</pre>
      85
                                                         и.
      86
                   (++myClock1).showTime();
      87
                  return 0;
      88
```

运算符重载的规则

- (1) C++ 几乎可以重载全部的运算符,而且只能够重载C++中已经有的。
 - 不能重载的运算符举例: "."、".*"、"::"、"?:"、"sizeof"
- (2) 重载之后运算符的优先级和结合性都不会改变。
- · (3)运算符重载是针对新类型数据的实际需要,对原有运算符进行适当的改造。

多学多问多思考



- 能否定义新的运算符
- 例如: ** 幂运算符





小结

- (1) 能够编码实现双目运算符的重载
- (2) 能够编码实现单目运算符的重载
- (3) 能够运用两种不同形式对运算符进行重载

延伸

请改编案例代码:对加号运算符进行重载,实 现时钟对象和整数的加法运算,整数和时钟对 象的加法运算。

请思考:插入运算符和提取运算符如何重载呢?