面向对象程序设计(OOP)

C++(第3章：类和对象)

福州大学·软件学院·软件工程系
王灿辉(wangcanhui@fzu.edu.cn)

友元函数和友元类

- 友元关系可以提高效率但破坏封装性。
- 友元关系无传递性、对称性和继承性

第3章：类和对象

友元函数

- 友元关系提供了不同类或对象的成员函数之间、类的成员函数和一般函数之间进行数据共享的机制。例：Point类和计算距离的函数。
- **可以提高效率但破坏封装性。**
- **友元关系无传递性、对称性和继承性**
- **友元函数与一般函数的调用方式和原理一样**
- 实例：Point(点)类和计算距离的友元函数

输出结果：

```
The distance is : 5
```

第3章：类和对象

友元函数

- 一个类的成员函数可以是另一个类的友元，但这时的友元说明前必须加上类名限定。

```
int func(int,float); //普通函数
class Ca{
  ……
public:
  void memfunc(char *);//类Ca的成员函数
  ……
};
```

第3章：类和对象

友元函数

- 定义普通函数func和类Ca的成员函数成为类Cb的友元函数：

```
class Cb{
    ……
public:
    friend int func(int,float);
    friend void Ca::memfunc(char *);
    ……
};
```

第3章：类和对象

友元类

- 例如，把类C1声明为类C2的友元类：

```
class  C2 {
    friend class C1; //类C1必须已经定义过
    …
};
```

- 这样，类C1的所有成员函数都是类C2的友元，即类C1的所有成员函数都可以访问类C2的私有(保护、公有)成员。

类的初始化参数表

- C++为类的构造函数提供了一种特殊的初始化类成员值的方式，而不用对其赋值。这就是通过成员初始化参数表，它用于对类的数据成员进行初始化，它放在构造函数的函数头后面，函数体的前面，形式为：

```
类名::类名(参数表):成员初始化参数表 {
    //类定义体
}
```

第3章：类和对象

类的初始化参数表

- 类的构造函数的一般定义格式为：

  类名(参数表0):成员1(参数表1),…,成员n(参数表n)

  {……}　其中，成员1…成员n是类中的对象数据成员，参数表1提供初始化成员1所需的参数，参数表n提供初始化成员n所需的参数，并且这几个参数表中的参数均来自参数表0，此外参数表0还必须包含初始化类的非对象成员所需的参数。

第3章：类和对象

类的初始化参数表

- 例如：

```
class Rectangle {
   int x1,y1;   //(x1,y1)是矩形右上角的坐标
   int h,w;     //h是矩形的高，w是矩形的宽
public:
   Rectangle(int,int,int,int);  //构造函数
   ......
};
Rectangle::Rectangle(int x=0,int y=0,int
height=0,int width=0)
:x1(x),y1(y),h(height),w(width){}
```

第3章：类和对象

类的初始化参数表

- 上述构造函数等价于：

```
Rectangle::Rectangle(int x=0,int y=0,int height=0,int width=0){
    x1=x;
    y1=y;
    h=height;
    w=width;
}
```

第3章：类和对象

类的初始化参数表

- 类的**引用成员和常成员**必须在成员初始化参数表中进行初始化。例如：


```
Myclass(int x,int y):ci(v),ri(i),rx(::x)
```

其中的ci为常成员，ri和rx为引用成员，i为普通私有成员，::x为全局变量

- **包含引用或常成员的类不能使用默认构造**
- 完整的实例

第3章：类和对象

一个类的对象作为另一个类的数据成员

- 例如：

```
class Point {……};
class Circle {
   Point centre;   //类的组合非常常用！
   ……
};
```

- 则在创建类Circle的对象(调用类Circle的构造函数)时，会自动调用类Point的构造函数。如果类Point的构造函数为有参函数时，一般使用初始化表的方式来调用构造函数。

第3章：类和对象

类的嵌套定义

- C允许定义**嵌套结构**(常用)。
- C++也允许定义嵌套类，但由于嵌套类使得类定义结构不清晰，所以一般很少使用，转而采用类分解(组合、继承等)更易理解。

第3章：类和对象

类的嵌套定义

- 例如：

```
class Date{……};
class Time{……};
class Schedule {
      int  number;  //序号
      Date date;    //自定义“日期”类
      Time time;    //自定义“时间”类
   public:
      ……
};                  //类的初始化实例
```

第3章：类和对象

类的组合

- 类的组合描述的就是一个类内嵌其他类的对象作为成员的情况，它们之间的关系是一种包含与被包含的关系。
- 构造函数的调用顺序：先调用内嵌对象的(缺省)构造函数(按对象在组合类中的声明顺序)，然后执行本类的构造函数的函数体。
- 可以在初始化表中初始化，并且比使用赋值语句初始化效率高。
- 析构函数的调用顺序与构造函数正好相反。
- 类的组合实例

前向引用声明

- C++规定类应该先先定义后使用，但当两个类相互引用时如何处理，例如：

```
#include <iostream>
class Ca {
public:
    void fa(Cb b);
    //编译错：error C2061: syntax error : identifier 'Cb'
};
class Cb {
public:
    void fb(Ca a);
};
void main(){}
```

第3章：类和对象

前向引用声明

- 解决办法就是采用前向引用声明，例如：

```
class Cb;  //前向引用声明
class Ca {
    Cb *pObj;                    //正确！
  public:
    Cb fa(Cb b1,Cb &b2); //正确！
    Cb &func(Cb *pObj);  //正确！
};
class Cb {
    void fb(Ca a);
};
```

- 前向引用声明的完整实例

运算符重载

- 运算符重载可以有三种方法：
  1、重载为类的成员函数
  2、重载为类的友元函数
  3、重载为普通函数

第3章：类和对象

为什么需要操作符重载？

- 例如：定义一个复数类，操作怎么办？
  是编写add、sub等成员函数？
  还是重载运算符+、-等？
- 显然如果允许重载操作符将使得复数的运算更为清晰，例如：

```
complex x(1,-2),y(0,3),z;
x=x+y; y=x-y; x=x-y; z=x+y;
```

第3章：类和对象

操作符（运算符）重载

- C++语言中预定义的运算符的操作对象只能是基本数据类型，但在实际应用中有许多用户自定义的类型也需要类似的功能，C++语言中的解决办法就是运算符重载。

第3章：类和对象

操作符（运算符）重载

- **运算符重载的实质是一种特殊的函数重载。**C++中的每一个运算符对应一个运算符函数。在实现过程中，把指定的运算符表达式中的运算符转化为对运算符函数的调用，而表达式中的运算对象转化为运算符函数的实参，这个过程是在**编译阶段完成**的。

第3章：类和对象

操作符（运算符）重载

- 例如：

```
int a=2,b=2;
a+b;
```


- 表达式“a+b”在编译时将被解析为函数调用形式：

```
operator+(a,b)
```

- 运算符重载可以有三种方法：
  1、重载为类的成员函数
  2、重载为类的友元函数
  3、重载为普通函数

第3章：类和对象

完整的一个类的定义

- 一个简单的矩形类实例
- 输出结果：

第1个矩形的面积为：6
第2个矩形的面积为：12
第3个矩形的面积为：6

相同对象的矩形相等

矩形右上角的坐标一样，矩形的高和宽相同，则认为矩形相等

否则认为矩形不相等

第3章：类和对象

运算符重载

- 在C++中，可以根据所创建的类的类型重载运算符。重载运算符的主要优点是允许将新数据类型无缝集成至编程环境中。
- 运算符重载的实例
- 输出结果：

```
Original value of a: 1, 2, 3
Original value of b: 10, 10, 10
Value of c after c = a + b: 11, 12, 13
Value of c after c = a + b + c: 22, 24, 26
Value of c after c = b = a: 1, 2, 3
Value of b after c = b = a: 1, 2, 3
```

第3章：类和对象

操作符（运算符）重载

- 除下列6个运算符外的运算符都可以被重载：“.”、“.*”、“::”、“?:”、typeid、sizeof
- **重载后的运算符不改变原有运算符的优先级、结合性、语法和参数个数。**
- 重载为类的成员函数时，函数的参数个数少1个(右++和--除外)；重载为友元函数时，参数个数与原运算符相同。
- **单目运算符最好重载为类的成员函数，而双目运算符则最好重载为类的友元函数。**

第3章：类和对象

操作符（运算符）重载

- 双目运算符重载为类的友元函数，则允许执行形如下述的语句(已经重载过+并且有单参构造函数)：

```
Complex c1(1,2),c2;
c2=c1+2.3;
c2=2.3+c1;
```

- 如果重载为成员函数，则c2=2.3+c1;将报告错误。

第3章：类和对象

操作符（运算符）重载

- 除重载()外，重载运算符至少需有一个参数的类型与自定义的类型有关，例如：
  float operator+(int,float);//错误
- 除函数调用运算符外，**重载的运算符不能有默认参数**，否则改变操作数的个数。
- 不能定义新的运算符（可改为定义新的函数）
- 最好在**重载运算符时保持它的原意**！！

第3章：类和对象

用友元函数进行运算符重载

- 在C++中，可以使用非成员函数为类重载运算符，它通常是类的友元。友元函数没有this指针，因此使用友元重载一元运算符必须显式传递两个操作数，重载二元运算符必须显式传递一个操作数。**不能用友元函数重载下述4个运算符：=、()、[]和->。而插入和抽取运算符<<和>>只能用友元函数重载**，不能定义为类的成员函数。
- 使用友元函数重载运算符的实例

第3章：类和对象

用友元函数进行运算符重载

- 在C++中，可以使用友元重载一元运算符++和—，但必须以引用传递参数。
- 根本不用的参数可以不予命名。
- 使用成员函数重载运算符++的实例
- 使用友元函数重载运算符--的实例
- 使用普通函数重载运算符--的实例

第3章：类和对象

用友元函数重载<<和>>运算符

- 为实现自定义类的输入和输出，可以重载插入和抽取运算符<<和>>，它们不能定义为类的成员函数，只能定义为一般的函数或类的友元函数。
- 使用友元函数重载运算符<<和>>
- 使用普通函数重载运算符<<和>>

第3章：类和对象

重载下标运算符[]

- 我们常用下标运算符operator[]来访问数组中的某个元素，它是一个双目运算符，第1个是数组名，第2个是数组下标。C++允许重载下标运算符operator[]，但只能重载为类的非静态(static)的成员函数。
- 重载下标运算符[]的实例

第3章：类和对象

操作符（运算符）重载

- 其他运算符，如：函数调用运算符()、递引用运算符->、内存分配和释放运算符new和delete等的重载实例参见其他书籍（麦中凡 编著，C++程序设计语言教程（语言基础），北京航空航天大学出版社，1995.）。

综合实例(运算符重载)

```
class Complex {
    double rpart;              //实部
    double ipart;              //虚部
  public:
    Complex operator-(); //重载运算符负号-
    friend Complex operator+(const Complex
   &c1,const Complex &c2);
    Complex operator+=(const Complex &c);
    ......
};
```

第3章：类和对象

操作符（运算符）重载：综合实例

```
//拷贝构造函数
Complex::Complex(const Complex &c)
{//与缺省拷贝构造函数一致！
  rpart=c.rpart;
  ipart=c.ipart;
}
```

第3章：类和对象

操作符（运算符）重载：综合实例

```
//（类型）转换构造函数
Complex::Complex(const double &r)
{
  rpart=r;
  ipart=0.0;
}
```

第3章：类和对象

操作符（运算符）重载：综合实例

```
//重载负号-运算符
Complex Complex::operator-()
{
  Complex result;
  result.rpart=-rpart;
  result.ipart=-ipart;
  return result;
}
```

第3章：类和对象

操作符（运算符）重载：综合实例

```
//重载算术运算符+
Complex operator+(const Complex
  &c1,const Complex &c2){//友元函数
  Complex result;
  result.rpart=c1.rpart+c2.rpart;
  result.ipart=c1.ipart+c2.ipart;
  return result;
}
```

第3章：类和对象

操作符（运算符）重载：综合实例

```
//重载算术运算符+=
Complex Complex::operator+=(const
  Complex &c) //重载为成员函数 {
  rpart+=c.rpart;
  ipart+=c.ipart;
  return *this;
}
```

第3章：类和对象

操作符（运算符）重载：综合实例

```
Complex c1;//缺省构造函数
Complex c2(7.0,9.0);//初始化构造函数
Complex c3(c2);//拷贝构造函数
//c3(c2)完全等价于c3=c2，均调用拷贝构造函数
//但不完全等价于Complex c3;c3=c2;它执行赋值操作
Complex c4(8.0);//(类型)转换构造函数
//c4(8.0)完全等价于c4=8.0，均调用(类型)转换构造函数
//但不完全等价于Complex c4;c4=8.0;它执行赋值操作
```

第3章：类和对象

操作符（运算符）重载：综合实例

```
c1=c3;      //赋值          c3=-c1;    //取负
c4=d;       //先进行隐式类型转换，再赋值！
c3=c1+c2; //双目加+
c3=c1+d;    //调用转换构造函数进行隐式类型转换
c3=d+c1;    //调用转换构造函数进行隐式类型转换
c3+=c2;
if (c1==c1) ......
if (c3==c1) ......
if (c4==8.0) ...... //先进行隐式类型转换后比较
```


- 综合实例：自定义复数类(摘略)

本部分内容讲授到此结束！

福州大学·软件学院·软件工程系
王灿辉(wangcanhui@fzu.edu.cn)