程序设计与算法（三）

C++面向对象程序设计

郭 炜

微信公众号

微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕!

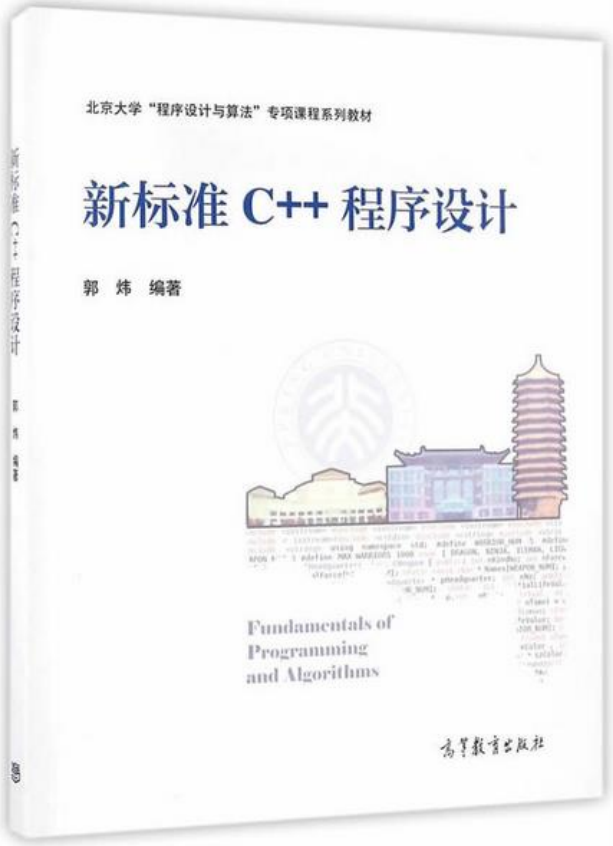
讲义照片均为郭炜拍摄

配套教材：

高等教育出版社

《新标准C++程序设计》

郭炜　编著



类和对象（1）

结构化程序设计

内蒙古锡盟草原

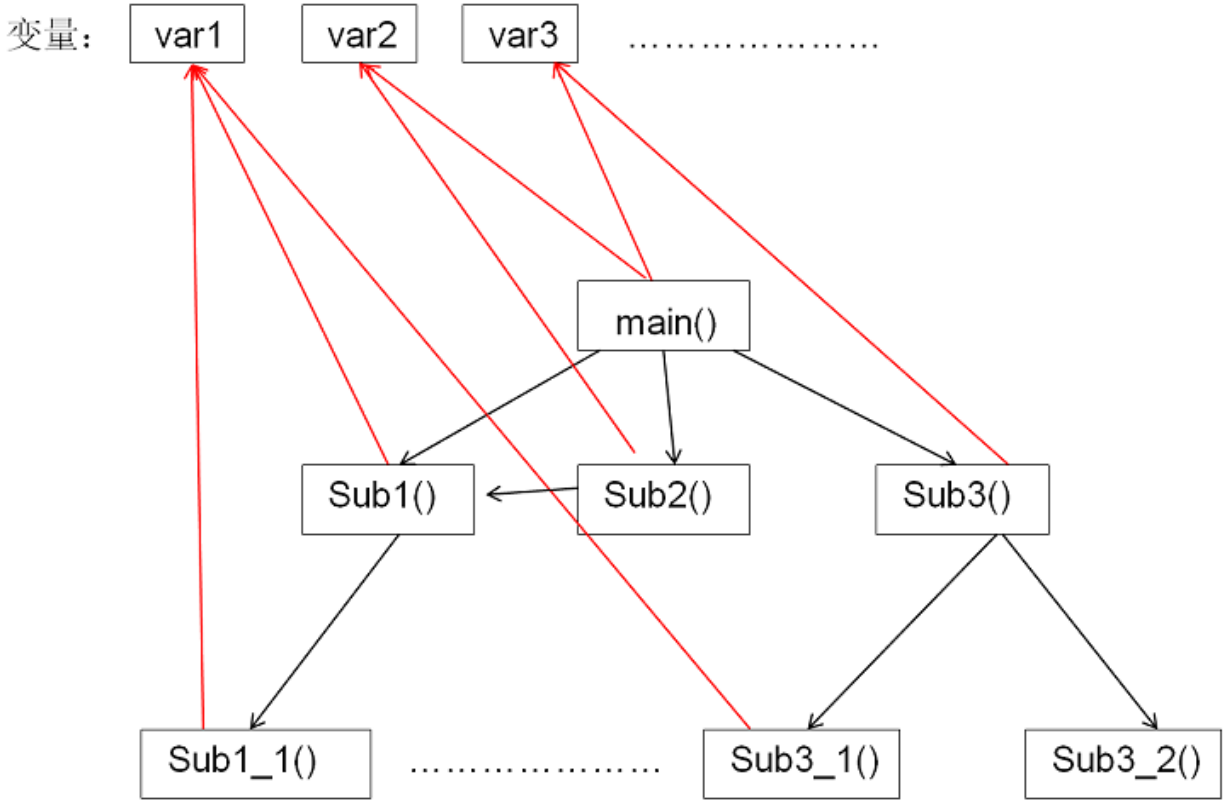
结构化程序设计

- C语言使用结构化程序设计：

  程序 = 数据结构 + 算法

  - 程序由全局变量以及众多相互调用的函数组成。
  - 算法以函数的形式实现，用于对数据结构进行操作。

结构化程序设计的程序模式：



结构化程序设计的不足

- 结构化程序设计中，函数和其所操作的数据结构，没有直观的联系。
- 随着程序规模的增加，程序逐渐难以理解，很难一下子看出来：
  - 某个数据结构到底有哪些函数可以对它进行操作？
  - 某个函数到底是用来操作哪些数据结构的？
  - 任何两个函数之间存在怎样的调用关系？

结构化程序设计的不足

- 结构化程序设计没有“封装”和“隐藏”的概念。要访问某个数据结构中的某个变量，就可以直接访问，那么当该变量的定义有改动的时候，就要把所有访问该变量的语句找出来修改，十分不利于程序的维护、扩充。
- 难以查错，当某个数据结构的值不正确时，难以找出到底是那个函数导致的。

结构化程序设计的不足

- 重用：在编写某个程序时，发现其需要的某项功能，在现有的某个程序里已经有了相同或类似的实现，那么自然希望能够将那部分代码抽取出来，在新程序中使用。

- 在结构化程序设计中，随着程序规模的增大，由于程序大量函数、变量之间的关系错综复杂，要抽取这部分代码，会变得十分困难。

结构化程序设计的不足

- 总之，结构化的程序，在规模庞大时，会变得难以理解，难以扩充（增加新功能），难以查错，难以重用。

结构化程序设计的不足

- 软件业的目标是更快、更正确、更经济地建立软件。
  - 如何更高效地实现函数的复用？
  - 如何更清晰的实现变量和函数的关系？使得程序更清晰更易于修改和维护。

面向对象程序设计

德国国王湖

面向对象的程序设计

- 面向对象的程序设计方法，能够较好解决上述问题。

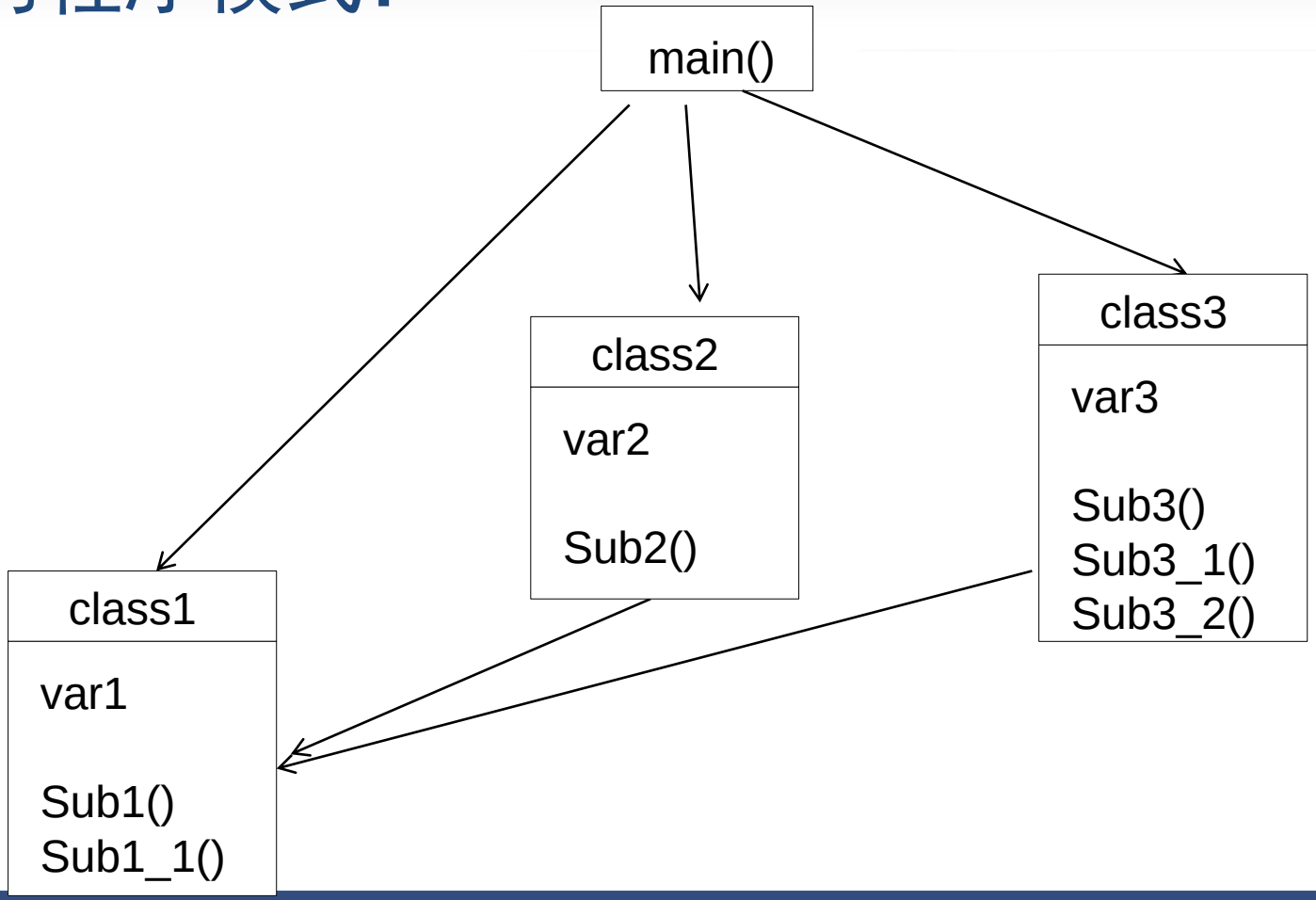
  面向对象的程序 = 类 + 类 + …+ 类

- 设计程序的过程，就是设计类的过程。

面向对象的程序设计

- 面向对象的程序设计方法：
  - 将某类客观事物共同特点（属性）归纳出来，形成一个数据结构（可以用多个变量描述事物的属性）；
  - 将这类事物所能进行的行为也归纳出来，形成一个个函数，这些函数可以用来操作数据结构(这一步叫“抽象”）。
- 然后，通过某种语法形式，将数据结构和操作该数据结构的函数“捆绑”在一起，形成一个“类”，从而使得数据结构和操作该数据结构的算法呈现出显而易见的紧密关系，这就是“封装”。
- 面向对象的程序设计具有“抽象”，“封装”“继承”“多态”四个基本特点。

面向对象的程序模式：


main()
class1
var1
Sub1()
Sub1_1()
class2
var2
Sub2()
class3
var3
Sub3()
Sub3_1()
Sub3_2()

类和对象

瑞士布里茨恩湖

从客观事物抽象出类

- 写一个程序，输入矩形的长和宽，输出面积和周长。
- 比如对于“矩形”这种东西，要用一个类来表示，该如何做“抽象”呢？
- 矩形的属性就是长和宽。因此需要两个变量，分别代表长和宽。
- 一个矩形，可以有哪些行为呢（或可以对矩形进行哪些操作）？
  - 矩形可以有设置长和宽，算面积，和算周长这三种行为（当然也可以有其他行为）。
  - 这三种行为，可以各用一个函数来实现，他们都需要用到长和宽这两个变量。

从客观事物抽象出类

- 将长、宽变量和设置长，宽，求面积，以及求周长的三个函数“封装”在一起，就能形成一个“矩形类”。

- 长、宽变量成为该“矩形类”的“成员变量”，三个函数成为该类的“成员函数”。成员变量和成员函数统称为类的成员。

- 实际上，“类”看上去就像“带函数的结构”。

从客观事物抽象出类

```
class CRectangle
{
      public:
            int w, h;
            int Area() {
                  return w * h;
            }
            int Perimeter(){
                  return 2 * ( w + h);
            }
            void Init( int w_,int h_ ) {
                  w = w_;  h = h_;
            }
}; //必须有分号
```

从客观事物抽象出类

```
int main( )
{
        int w,h;
        CRectangle  r;   //r是一个对象
        cin >> w >> h;
        r.Init( w,h);
        cout << r.Area() << endl <<
                r.Perimeter();
        return 0;
}
```

从客观事物抽象出类

➢通过类，可以定义变量。类定义出来的变量，也称为类的实例，就是我们所说的“对象” 。

➢C++中，类的名字就是用户自定义的类型的名字。可以象使用基本类型那样来使用它。CRectangle 就是一种用户自定义的类型。

对象的内存分配

- 和结构变量一样，对象所占用的内存空间的大小，等于所有成员变量的大小之和。
  对于上面的CRectangle类，sizeof(CRectangle) = 8

- 每个对象各有自己的存储空间。一个对象的某个成员变量被改变了，不会影响到另一个对象。

对象间的运算

和结构变量一样，对象之间可以用“=”进行赋值，但是不能用“==”，“!=”，“>”，“<”“>=”“<=”进行比较，除非这些运算符经过了“重载”。

使用类的成员变量和成员函数

用法1：对象名.成员名

```
CRectangle r1,r2;
r1.w = 5;
r2.Init(5,4);
```

- Init函数作用在 r2 上，即Init函数执行期间访问的 w 和 h是属于 r2 这个对象的，执行r2.Init 不会影响到 r1。

使用类的成员变量和成员函数

用法2.　　指针->成员名

```
CRectangle  r1,r2;
CRectangle * p1 = & r1;
CRectangle * p2 = & r2;
p1->w = 5;
p2->Init(5,4);    //Init作用在p2指向的对象上
```

使用类的成员变量和成员函数

用法3：引用名.成员名

```
CRectangle r2;
CRectangle & rr = r2;
rr.w = 5;
rr.Init(5,4);        //rr的值变了，r2的值也变

void PrintRectangle(CRectangle & r)
{  cout << r.Area() << ","<<  r.Perimeter(); }
CRectangle r3;
r3.Init(5,4);
PrintRectangle(r3);
```