第8章　类和对象

- 欢迎开启面向对象程序设计的大门，在第1章中已经简要介绍过面向对象设计的基本理念（见备注），本章将具体讲述类和对象的概念。在前面的章节中，程序是由一个个函数组成的，是结构化的面向过程的编程方法。从本章开始，编写的程序是由对象组成的，将要学习用C++语言进行面向对象的程序设计，当然，面向对象设计也离不开函数等前面讲述的基础知识。
- C++的四大特性：抽象、封装、继承、多态
- 源文件名由*.c改成*.cpp(CPP=C Plus Plus=C++)
- 用结构定义变量时，不再需要struct前缀。

8.1　面向对象基本概念

- “对象”（object）是个抽象的概念，现实世界中的任何事物都可以看成是对象，动物、植物、摩托车、汽车等等都是对象，对象之间有很大的差异，如人和汽车，但有的对象间有相似之处，比如摩托车和自行车，它们有共同的特征（有轮子），同样的功能（人的交通工具），也有不同的特征，如“轮子个数”，“车子重量”等等，基于此，可将“有轮子”，“可更换轮胎”、“能作为人的交通工具”抽象成一个类别（class），可称之为“车”类，摩托车和自行车是该类别的对象。
- 类的提取往往是从两个方面来考虑的，一是特征（C++常称为“属性”）、另一个是功能（C++中常称为“行为”），具备类中定义的“属性”和“行为”的对象都是该类的对象，因此，我们可以说，电动车也是“车”类的对象。

8.1.1　类的概念

- C++用类来描述对象，类是对现实世界中相似事物的抽象，同是“双轮车”的摩托车和自行车，有共同点，也有许多不同点。“车”类是对摩托车、自行车、汽车等相同点和不同点的提取与抽象，如所示。
- 类的定义分为两个部分：数据（相当于属性）和对数据的操作（相当于行为）。从程序设计的观点来说，类就是数据类型，是用户定义的数据类型，对象可以看成某个类的实例（某个类的变量），类和对象的关系与前面介绍的“结构”和“结构体变量”的关系相似，但又有不同，在本章稍后类的定义一节中后具体说明这一问题。

图 8.1　“车”类示意图

8.1.2　类是分层的

- 每一大类中可分成若干小类，也就是说，类是分层的，如图8.2所示。可将所有的图形抽象成“图形”类，该类中共同的属性有很多，这里只取“颜色”这个属性，对所有图形而言，都可定义“显示”操作。同时，“图形”类可进一步分为“一维图形”类、“二维图形”类和其他类，根据形状的不同，“一维图形”类可进一步分为“直线”类和“折线”类，“二维图形”类又可分为“正方形”类和“圆”类。下层的类除了“继承”了上层类中定义的属性和行为外，还可增加新的属性和行为（如“圆”类相比“二维图形”类增加了“圆心”和“半径”属性，增加了“求面积”这一行为），甚至可以在下层类中重新定义上层类已定义的属性和行为（如“直线”类、“折线类”、“正方形”类和“圆”类中都重新定义了“图形”类中已定义的“显示”操作）。

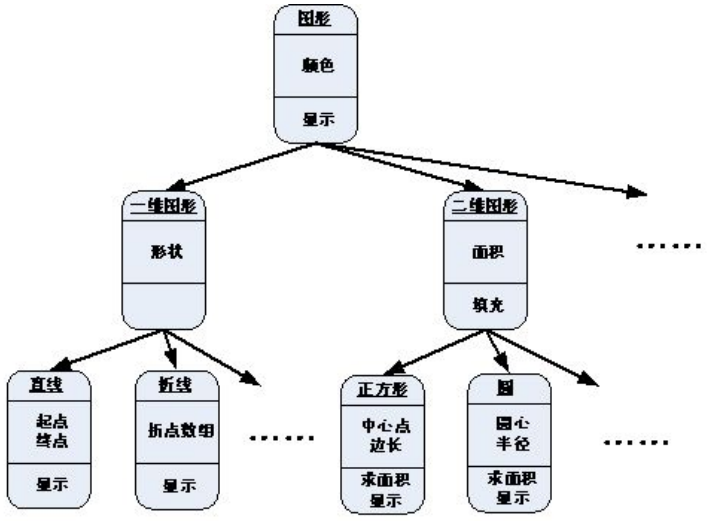

图8.2　类是分层次的

8.1.3　类和对象的关系

对象需要从属性和行为两个方面进行描述，类是对象的封装。类的使用主要有以下几个步骤：

- 定义类：C++中，分别用数据成员和函数成员来表现对象的属性和行为。类的定义强调“信息隐藏”，将实现细节和不允许外部随意访问的部分屏蔽起来。因此，在类定义中，需要用public或private将类成员区分开（此外，还有protected型的数据成员，后面课程会有介绍），外界不能访问程序的private成员，只能访问public数据成员，对象间的信息传送也只能通过public成员函数，保证了对象的数据安全。
- 实现类：即进一步定义类的成员函数，使各个成员函数相互配合以实现接口对外提供的功能，类的定义和实现是由类设计者完成的。
- 使用类：通过该类声明一个属于该类的变量（即对象），并调用其接口（即public型的数据成员或函数成员），这是使用者的工作。

8.2　C++类的定义

- 先来看一下类是如何定义的，对一些通用的问题，前人已经定义好了很多的类，比如微软的MFC类库，程序员不必关心其内部细节，只要抱着“拿来主义”的态度就好，但对某些特殊问题来说，必须由自己提炼模型，进行类的定义。

8.2.1　类定义的基本形式

C++中使用关键字class定义一个类，其基本形式如下：

```
class 类名
{
private:
   私有成员变量和函数
protected:
   保护成员变量和函数
public:
   公共成员变量和函数
}; //不要漏写了这个分号;
```

8.2.2　类定义示例

- 对一台计算机来说，它有如下特征：
- 属性：品牌、价格。
- 方法：输出计算机的属性。
- 如下代码8-1实现了computer类的定义：

```
//文件example801.h
class computer
{
private:          //私有成员列表，这里的private可以省略
   char brand[20];
   float price; //不能在这里初始化，如float price=0是错误的
public:           //公共成员列表（接口）
   void print();
   void SetBrand(char* sz);
   void SetPrice(float pr);
};
```

8.2.3 class和struct

- class的定义看上去很像struct定义的扩展，事实上，类定义时的关键字class完全可以替换成struct，也就是说，第5章中介绍的结构体变量也可以有成员函数。class和struct的唯一区别在于：struct的默认访问方式是public，而class为private。
- 提示：通常使用class来定义类，而把struct用于只表示数据对象、没有成员函数的类。

8.3　C++类的实现

- 类的实现就是定义其成员函数的过程，类的实现有两种方式：
  - 在类定义时同时完成成员函数的定义。
  - 在类定义的外部定义其成员函数。

8.3.1　在类定义时定义成员函数

- 成员函数的实现可以在类定义时同时完成
- 参照备注代码8-2

8.3.2　在类定义的外部定义成员函数

在类定义的外部定义成员函数时，应使用作用域操作符（::）来标识函数所属的类，即有如下形式：

```
返回类型 类名::成员函数名(参数列表)
{
   函数体
}
```

其中，返回类型、成员函数名和参数列表必须与类定义时的函数原型一致。代码见备注8-3。

8.4　C++类的使用

- 定义了一个类之后，便可以如同用int、double等类型符声明简单变量一样，创建该类的对象，称为类的实例化。由此看来，类的定义实际上是定义了一种类型，类不接收或存储具体的值，只作为生成具体对象的“蓝图”，只有将类实例化，创建对象（声明类的变量）后，系统才为对象分配存储空间。

8.4.1　声明一个对象

- 备注代码8-4使用类定义声明了一个对象，并利用对象名实现了public成员函数的调用

8.4.2　对象的作用域、可见域和生存周期

- 对象的作用域、可见域和生存期与普通变量，如int型变量的作用域、可见域和生存周期并无不同，对象同样有局部、全局和类内（稍后就将对对象成员进行介绍）之分，对于在代码块中声明的局部对象，在代码块执行结束退出时，对象会被自动撤销，对应的内存会自动释放（当然，如果对象的成员函数中使用了new或malloc申请了动态内存，却没有使用delete或free命令释放，对象撤销时，这部分动态内存不会自动释放，造成内存泄露）。
- 跟踪调试，查看同一个类的不同对象的成员变量和成员函数在内存中的地址分配情况。结论：成员变量占据不同的内存区域(堆、栈)；成员函数共用同一内存区域(代码段)。

练习1

东西不多，理解去写，
20分钟

- 1、创建文件student.h，并在文件中声明类

- 2、定义一个类，类中分别有private、protected、public类型的成员数据和成员函数，并在成员函数内部和类的外部测试各种成员数据和成员函数的可访问性。

8.5. 对象的创建和撤销

- 代码8.4中，通过自定义的公共成员函数SetBrand和SetPrice实现对数据成员的初始化，实际上，C++为类提供了两种特殊的成员函数来完成同样的工作：
  1. 一是构造函数，在对象创建时自动调用，用以完成对象成员变量等的初始化及其他操作（如为指针成员动态申请内存空间等）；如果程序员没有显式的定义它，系统会提供一个默认的构造函数。
  2. 另一个是析构函数，在对象撤销时自动调用，用以执行一些清理任务，如释放成员函数中动态申请的内存等。如果程序员没有显式的定义它，系统也会提供一个默认的析构函数。

8.5.1 构造函数的作用

有例子

- 当对象被创建时，构造函数自动被调用。构造函数有一些独特的地方：1、函数的名字与类名相同，2、没有返回类型和返回值，即使是void也不能有。其主要工作有：
  - 给对象一个标识符。
  - 为对象数据成员开辟内存空间。
  - 完成对象数据成员的初始化（在函数体内进行，由程序员完成）。
- 上述3点也说明了构造函数的执行顺序，在执行函数体之前，构造函数已经为对象的数据成员开辟了内存空间，这时，在函数体内对数据成员的初始化便是顺理成章了。
- 备注代码给出了point类的显式构造函数

8.5.2　构造函数可以有参数

有例子

- 编译器自动生成的缺省构造函数是无参的，实际上，构造函数可以接收参数，在对象创建时提供更大的自由度，如备注代码8-5
- 一旦用户定义了构造函数，系统便不再提供默认构造函数。
- 跟踪执行，理解构造函数的执行顺序。

8.5.3　构造函数支持重载

- 一旦程序员为一个类定义了构造函数，编译器便不会为类自动生成缺省构造函数，因此，如果还想使用无参的构造函数，如“point pt0;”的形式必须在类定义中显式定义一个无参构造函数。这样，构造函数就会出现两个，会不会有问题呢？不会，构造函数支持重载，在创建对象时，根据传递的具体参数决定采用哪个构造函数。
- 见备注代码8-6

8.5.4　构造函数允许按参数缺省方式调用

- 代码8-5中的构造函数可以作如下定义：

```
point(int x＝0,int y＝0)
{
    cout<<"对象创建时构造函数被自动调用"<<endl;
    xPos=x;
    yPos=y;
}
```

- 此时，可在创建对象时省略参数，下列语句都是合法的：

```
point pt;       //x和y都采用默认值0
point pt(3);    //x为3，y采用默认值0
point pt(3,4);//x为3，y为4，两个参数都不采用默认值
```

8.5.5　初始化表达式1

- 除了在构造函数体内初始化数据成员外，还可以通过成员初始化表达式来完成。成员初始化表达式可用于初始化类的任意数据成员（后面要介绍的static数据成员除外），该表达式由逗号分隔的数据成员表组成，初值放在一对圆括号中。只要将成员初始化表达式放在构造函数的头和体之间，并用冒号将其与构造函数的头分隔开，便可实现数据成员表中元素的初始化，对代码8-6而言，下述代码：

```
point(int x,int y)
{
    cout<<"有参构造函数的调用"<<endl;
    xPos=x;
    yPos=y;
}
```

等价于：

```
point(int x,int y):xPos(x),yPos(y)
{
   cout<<"有参构造函数的调用"<<endl;
}
```

相当于赋值

8.5.5　初始化表达式2

初始化成员列表以冒号开头，后跟一系列的以逗号分隔开来的初始化字段

- 每个成员在初始化表中只能出现一次
- 初始化的顺序不是由成员变量在初始化表中的顺序决定的，而是由成员变量在类中被申明时的顺序决定的。理解这一点有助于避免意想不到的错误。见备注代码8-7
- 练习：
  - 1、写程序，测试在初始化成员列表各个成员初始化的顺序。结论：初始化的顺序和声明时的顺序一致。
  - 2、写程序，测试在初始化成员列表中和在构造函数体中赋值两者的先后顺序。结论：初始化成员列表的赋值语句先执行，构造函数体中的赋值语句后执行。

8.5.6 析构函数

malloc对应free

new对应delete

有例子

- 构造函数在创建对象时被系统调用，而析构函数在对象被撤销时被自动调用，相比构造函数，析构函数要简单的多。析构函数有如下特点：
  - 与类同名，之前冠以波浪号，以区别于构造函数。
  - 析构函数没有返回类型，也不能指定参数，因此，析构函数只能有一个，不能被重载。
  - 对象超出其作用域被销毁时，析构函数会被自动调用。
- 如果用户没有显式地定义析构函数，编译器将为类生成“缺省析构函数”，缺省析构函数是个空的函数体，只清除类的数据成员所占据的空间，但对类的成员变量通过new和malloc动态申请的内存无能为力，因此，对于动态申请的内存，应在类的析构函数中通过delete或free进行释放，这样能有效避免对象撤销造成的内存泄漏。见代码8-8

8.5.7　显式调用析构函数

- 程序员不能显式调用构造函数，但却可以调用析构函数控制对象的撤销，释放对象所占据的内存空间，以更高效地利用内存，如：

```
#include "computer.h"
int main()
{
    computer comp("Dell",7000);
    comp.print();
    comp.~computer();//显式调用析构函数，comp被撤销
    return 0;
} //注意这里comp对象释放时析构函数也被隐式调用了一次
```

- 虽然可以显式调用析构函数，但不推荐这样做，因为可能带来重复释放指针类型变量所指向的内存等问题。完整代码见备注

8.6. 复制构造函数

- C++中经常使用一个常量或变量初始化另一个变量，例如：

```
double x＝5.0;
double y＝x;
```

- 使用类创建对象时，构造函数被自动调用以完成对象的初始化，那么能否象简单变量的初始化一样，直接用一个对象来初始化另一个对象呢？答案是肯定的，以point类为例：

```
point pt1(2,3);
point pt2=pt1;
```

- 后一个语句也可写成：

```
point pt2( pt1);
```

- 上述语句用pt1初始化pt2，相当于将pt1中每个数据成员的值复制到pt2中，这是表面现象。实际上，系统调用了一个复制构造函数。如果类定义中没有显式定义该复制构造函数时，编译器会隐式定义一个缺省的复制构造函数，它是一个inline、public的成员函数，其原型形式为：

```
point:: point (const point &);
```

8.6.1　复制构造函数调用机制

- 复制构造函数的调用示例：
  - `point pt1(3,4);　　//构造函数`
  - `point pt2(pt1);　　//复制构造函数`
  - `point pt3 = pt1;　　//复制构造函数`
- 见代码8-9

8.6.2　缺省复制构造函数带来的问题

- 缺省的复制构造函数并非万金油，在一些情况下，必须由程序员显式定义缺省复制构造函数，先来看一段错误代码示例，见备注代码8-10。
- 其中语句computer comp2(comp1)等价于：

```
comp2.brand = comp1.brand;
comp2.price = comp1.price;
```

- 后一句没有问题，但comp2.brand = comp1.brand却有问题：经过这样赋值后，两个对象的brand指针都指向了同一块内存，当两个对象释放时，其析构函数都要delete[]同一内存块，便造成了2次delete[]，从而引发了错误。

8.6.3　解决方案——显式定义复制构造函数

- 如果类中含有指针型的数据成员、需要使用动态内存，程序员最好显式定义自己的复制构造函数，避免各种可能出现的内存错误，见代码8-11。

```
computer(const computer &cp) //自定义复制构造函数
{
    //重新为brand开辟和cp.brand同等大小的动态内存
    brand = new char[strlen(cp.brand) + 1];
    strcpy(brand, cp.brand); //字符串复制
    price = cp.price;
}
```

8.6.4　关于构造函数和复制构造函数

- 复制构造函数可以看成是一种特殊的构造函数，这里姑且区分为“复制构造函数”和“普通构造函数”，因此，它也支持初始化表达式。
- 创建对象时，只有一个构造函数会被系统自动调用，具体调用哪个取决于创建对象时的参数和调用方式。C++对编译器何时提供缺省构造函数和缺省复制构造函数有着独特的规定，如表8.1所示。

表 8.1　系统何时提供缺省构造函数和缺省复制构造函数

用户是否自定义了一个普通构造函数	用户是否自定义了复制构造函数	编译器是否自动提供缺省构造函数	编译器是否自动提供缺省复制构造函数
否	否	是	是
否	是	否	否
是	否	否	是
是	是	否	否

8.6.5 构造函数调用示例

- CTest t0();　　//这是函数的声明，不是实例化类
- CTest t1;　　//缺省构造函数
- CTest t2(1);　　//一个参数的构造函数
- CTest t3(1, 2);　　//两个参数的构造函数
- CTest t4 = 1;　　//等价于CTest t4(1); //explicit
- CTest t5 = t1;　　//CTest(t1);
- CTest t6 = CTest();//CTest(1); CTest(1,2);
- t6 = CTest(1);
- t6 = 1;　　//首先调用单个参数的构造函数，生成临时
  //对象CTest(1)，然后调用赋值运算符函数
- t6 = t1;　　//调用赋值运算符函数
- 见备注代码。请注意输出的地址值，观察构造函数和析构函数的配对情况。

8.7　特殊数据成员的初始化

- 有4类特殊的数据成员（常量成员、引用成员、类对象成员、静态成员），其初始化及使用方式与前面介绍的普通数据数据成员有所不同，下面展开具体讨论。

8.7.1　const数据成员的初始化

- 数据成员可以由const修饰，这样，一经初始化，该数据成员便具有“只读属性”，在程序中无法对其值修改。
- 事实上，在构造函数体内或复制构造函数体内初始化const数据成员是非法的，如代码8-12
- const数据成员只能通过成员初始化表达式进行初始化，如代码8-13

8.7.2　引用成员的初始化

- 对于引用类型的数据成员，同样只能通过成员初始化表达式进行初始化，见代码8-14

8.7.3　类对象成员的初始化

- 类数据成员也可以是另一个类的对象，比如，一个直线类对象中包含两个point类对象，在直线类对象创建时可以在初始化列表中初始化两个point对象，当然也可以在构造函数里初始化它们。如下：
- line(int x1, int y1, int x2, int y2):pt1(x1, y1), pt2(x2, y2) //line对象的有参构造函数
- {
- //　pt1 = point(x1, y1);//如果不在初始化表达式里初始化，也可以这样初始化
- //　pt2 = point(x2, y2);
- 　　cout << "线的构造函数被执行" << endl;
- }
- 代码8-15中是对直线类和point类的实现

8.7.4　特别说明

- 对复制构造函数来说，一旦给出了自己定义的形式，编译器便不会提供缺省的复制构造函数，因此，确保自定义的复制构造函数的有效性很重要。因此，在一些必须使用自定义复制构造函数的场合，掌握特殊成员的用法很必要。所举例子中，尽管有些复制构造函数纯属“画蛇添足”，用系统提供的缺省复制构造函数足以实现想要的功能，但还是给出了完整的书写形式，这就是原因所在。

8.7.5 static数据成员的初始化

- C++允许使用static（静态存储）修饰数据成员，这样的成员在编译时就被创建并初始化的（与之相比，对象是在运行时被创建的），且其实例只有一个，被所有该类的对象共享，就像住在同一宿舍里的同学共享一个房间号一样。静态数据成员和第6章中介绍的静态变量一样，程序执行时，该成员已经存在，一直到程序结束，任何对象都可对其进行访问。
- 静态数据成员的初始化必须在类申明之外进行，且不再包含static关键字，格式如下：

  类型 类名::变量名 = 初始化表达式; //普通变量

  类型 类名::对象名(构造参数); //对象变量
- 如float computer::total_price = 0;
- 代码见8-17

8.8　特殊函数成员

- 除了构造函数、复制构造函数和析构函数外，其他成员函数被用来提供特定的功能，一般来说，提供给外部访问的函数称为接口，访问权限为public，而一些不供外部访问，仅仅作为内部功能实现的函数，访问权限设为private。本节主要讨论函数成员的一些特殊用法。

8.8.1　静态成员函数

- 成员函数也可以定义成静态的，与静态成员变量一样，系统对每个类只建立一个函数实体，该实体为该类的所有对象共享。
- 静态成员函数体内不能使用非静态的成员变量和非静态的成员函数。
- 静态成员函数用法示例见代码8-18

8.8.2 const与成员函数

- 第7章已经介绍了const在函数中的应用，实际上，const在类成员函数中还有种特殊的用法，把const关键字放在函数的参数表和函数体之间（与第7章介绍的const放在函数前修饰返回值不同），称为const成员函数，其特点有二：
  1. 只能读取类数据成员，而不能修改之
  2. 只能调用const成员函数，不能调用非const成员函数
- 其基本定义格式为：
- （1）类内定义时：

```
类型 函数名(参数列表) const
{
    函数体
}
```

- （2）类外定义时，共分两步：
- 类内声明：

```
类型 函数名(参数列表) const;
```

- 类外定义

```
类型 类名::函数名(参数列表) const
{
    函数体
}
```

- 见代码8-19

8.9　对象的组织

- 有了自己定义的类，或者使用别人定义好的类创建对象，其机制与使用int等创建普通变量几乎完全一致，同样可以const对象、创建指向对象的指针，创建对象数组，还可使用new和delete等创建动态对象。

8.9.1　const对象和const对象引用

- 类对象也可以声明为const对象，一般来说，能作用于const对象和const对象引用的成员函数除了构造函数和析构函数，便只有const成员函数了，因为const对象只能被创建、撤销以及只读访问，改写是不允许的。
- 同样的，const对象的引用也只能调用const成员函数，以及构造函数和析构函数。
- 见代码8-20

8.9.2　指向对象的指针

- 对象占据一定的内存空间，和普通变量一致，C++程序中采用如下形式声明指向对象的指针：
- 类名*　指针名 [=初始化表达式];
- 初始化表达式是可选的，既可以通过取地址（&对象名）给指针初始化，也可以通过申请动态内存给指针初始化，或者干脆不初始化（比如置为NULL），在程序中再对该指针赋值。
- 指针中存储的是对象所占内存空间的首地址。
- 定义point类如备注中的代码
- 针对上述定义，则下列形式都是合法的：
  - point pt;　　//默认构造函数
  - point *ptr = NULL;　　//空指针
  - point *ptr = &pt;　　//取某个对象的地址
  - point *ptr = new point(1, 2.2);//动态分配内存并初始化
  - point *ptr = new point[5];　　//动态分配5个对象的数组空间
- 使用指针对象：ptr->print();(*ptr).print();都是合法的。

8.9.3 对象的大小(sizeof)

- 对象占据一定大小的内存空间。总的来说，对象在内存中是以结构形式（只包括非static数据成员）存储在数据段或堆中，类对象的大小（sizeof）一般是类中所有非static成员的大小之和。在程序编译期间，就已经为static变量在静态存储区域分配了内存空间，并且这块内存在程序的整个运行期间都存在。而类中的成员函数存在于代码段中，不管多少个对象都只有一个副本。
- 对象的大小同样遵循第5章中介绍的3条准则，但有一些特殊之处需要强调：
  - C++将类中的引用成员当成“指针”来维护，占据4个内存字节。
  - 如果类中有虚函数(后面课程将会介绍)时，虚析构函数除外，还会额外分配一个指针用来指向虚函数表（vtable），因此，这个时候对象的大小还要加4。
  - 指针成员和引用成员属于“最宽基本数据类型”的考虑范畴。
- 见代码8-21

8.9.4 this指针

- 前面提到，一个类的所有对象共用成员函数代码段，不管有多少个对象，每个成员函数在内存中只有一个版本，那编译器是如何知道是哪个对象在执行操作呢，答案就是“this指针”。
- this指针是隐含在成员函数内的一种指针，称为指向本对象的指针，可以采用诸如“this->数据成员”的方式来存取类数据成员。
- 举例来说：

```
class Ex
{
private:
     int x;
     int y;
public:
     void Set()
     {
          this->x=1;
          this->y=2;
     }
};
```

8.9.5　对象数组

- 对象数组和标准类型数组的使用方法并没有什么不同，也有声明、初始化和使用3个步骤。

- (1)对象数组的声明：类名 数组名[对象个数];
  这种格式会自动调用无参或所有参数都有缺省值的构造函数，类定义要符合该要求，否则编译报错。

- (2)对象数组的初始化：可以在声明时初始化。
  对于point(int ix,int iy){}这种没有缺省参数值的构造函数：

```
point pt[2]={point(1,2), point(3,4)};    //#1 正确
point pt[ ]={point(1,2), point(3,4)};    //#2 正确
point pt[5]={point(1,2), point(3,4)};    //#3 错误
```

  语句#1和#2是正确的，但语句#3错误，因为pt的后3个元素会自动调用无参的或者所有参数都有缺省值的构造函数，但这样的构造函数不存在。

- 见备注代码

8.9.6　对象链表

- 对象链表中，节点的初始化需要构造函数来完成，除此之外，对象链表和C语言中介绍的链表并无不同。

8.10　为对象动态分配内存

- 和把一个简单变量创建在动态存储区一样，可以用new和delete为对象分配动态存储区，在复制构造函数一节中已经介绍了为类内的指针成员分配动态内存的相关范例，本节主要讨论如何为对象和对象数组动态分配内存。

8.10.1　使用new和delete为单个对象分配/释放动态内存

- ```
  int main()
  {
      point* p = new point(4,5);
      //动态申请一块内存，存储point类对象，并将地址赋值给
      point型指针p
      p->print();//使用指针加－>调用成员函数
      delete p; //释放动态申请的内存，防止内存泄露
      p=NULL;   //养成良好习惯，防止野指针
      return 0;
   }
  ```

- 详见代码8-22
  ```

8.10.2　使用new和delete[]为对象数组分配/释放动态空间

- 使用new为对象数组分配动态空间时，不能显式指定对象调用哪个构造函数，因此，对象要么没有定义任何形式的构造函数（由编译器缺省提供），要么显式定义了一个（且只能有一个）所有参数都有缺省值的构造函数。
- 见代码8-23

8.10.3 malloc和free不能为对象动态申请内存

- malloc/free无法满足动态对象的要求，因为malloc和free无法像new/delete及new/delete[]那样自动调用对象的构造函数和析构函数。

8.11　小结

- 本章讲述了C++语言中面向对象编程的基本概念和方法。
- C++通过class关键字可以定义类，类的成员包括数据成员和函数成员两种。关于类的使用，大体分为类的定义、类的实现和类对象的创建3个步骤，其中，类的定义指明了类的结构，相当于“蓝图”，而类的实现相当于“技术图纸”，根据定义和实现便可以声明一个类的对象。
- 类中有几个特殊的成员函数，构造函数、复制构造函数和析构函数。构造函数和复制构造函数用于为类对象开辟所需内存空间，并初始化各成员变量的值，而析构函数则是在撤销对象时，释放其内存空间，但需要注意的是，用户通过new申请的动态内存并不会在对象撤销时被自动释放，所以，应合理搭配new和delete，及时释放无用的动态内存。构造函数不能由用户调用，但析构函数可以显式调用。复制构造函数的形参是本类对象的引用，它是用一个对象来初始化另一个对象。如果编程者没有显式定义构造函数（包括复制构造函数），C++编译器就隐式定义缺省的构造函数。

作业：

1. 定义一个满足如下要求的Date类
   a) 用下面的格式输出日期
      年.月.日 (2011.01.31)
   b) 可以实现在日期上加一天操作(不要用运算符重载)
   c) 可以实现设置日期、打印日期操作
   d) 年,月,日可以各用一个整数表示