第 6 章　理解面向对象编程

类扩展了 C++内置的功能，可帮助用户表示并解决复杂的实际问题。

本章将学习：

- 什么是类和对象？
- 如何定义新类并创建这个类的对象？
- 什么是成员函数和成员数据？
- 什么是构造函数？如何使用它们？

6.1　C++是面向对象的吗

C++的前身 C 语言一度是世界上最流行的用于商业软件开发的程序设计语言，它被用于开发操作系统（如 UNIX 操作系统）、进行实时编程（机器、设备和电子控制），随后才被用作一种常规编程语言。其目标是提供一种更容易、更安全的与硬件交互的编程方式。

C 是作为一种介于高级商业应用程序语言（如 COBOL）与低级、高性能、很难使用的汇编语言之间的中间语言而开发的。C 语言采用结构化编程，在这种编程方式中，问题被分解成较小的名为过程的重复活动单元，而数据被组织成名为结构的包。

然而，诸如 Smalltalk 和 CLU 等研究用语言开拓了一个新方向：面向对象，将存储在结构中的数据与过程的功能组合成一个单元：对象。

现实世界中充斥着对象：汽车、狗、树、云、花。每个对象都有其特征（速度快、友好、棕色、蓬松、可爱）；大多数对象都有行为（移动、叫、长大、下雨、枯萎）。通常，人们不会考虑汽车的规格以及如何操纵这些规格，而是将汽车视为一种有特定外观和行为的对象。在计算机领域描述现实世界中的对象时，情况也如此。

我们在 21 世纪初编写的程序要比在 20 世纪末编写的程序复杂得多。使用过程性语言编写的程序通常难以管理和维护，且扩展的费用高昂。图形用户界面、Internet、数字和无线电话以及大量新技术的出现，极大地增加了项目的复杂性，同时用户对用户界面质量的期望也在增加。

面向对象软件开发提供了帮助应对这种软件开发挑战的工具。虽然对于复杂的软件开发，并不存在什么灵丹妙药，但面向对象编程语言将数据结构和操纵数据的方法紧密地联系在一起，更符合人类（程序员和客户）的思维方式，可促进交流，改善交付的软件的质量。在面向对象编程中，考虑的不再是数据结构和操纵它们的函数，而是对象，它们就像现实世界中的物体：有特定的外观和行为。

创建 C++旨在面向对象编程和 C 语言之间架起一座桥梁，其目标是提供一种面向对象设计的快速商用软件开发平台，并将重点放在高性能上。接下来读者将更详细地了解到 C++是如何实现其目标的。

6.2　创建新类型

编写程序通常旨在解决实际问题，如跟踪员工记录或模拟供暖系统的工作原理。虽然只使用数字和字符

编写的程序也可以解决复杂的问题，但如果能够创建你谈论的对象，则解决复杂的大型问题将容易得多。换句话说，如果能创建表示房间、热传感器、温度调节装置和锅炉的变量，则模拟供热系统工作原理将容易得多。这些变量与现实情况联系越紧密，程序编写起来越容易。

前面介绍了大量的变量类型，包括 unsigned int 变量和 char 变量。类型指出了有关变量的很多信息。例如，如果 Height 和 Width 被声明为 unsigned short 变量，你将知道其中每个变量都可以存储一个 0～65535 的数，这里假设 unsigned short 占用 2 字节。这就是 unsigned short 的含义，如果在这些变量中存储其他的数据，将导致错误。不能将你的姓名存储在 unsigned short 变量中，也不应试图这样做。

通过将 Height 和 Width 声明为 unsigned short，你知道可以将它们相加，并将结果赋给另一个数值变量。

变量类型提供了如下信息：

- 变量在内存中的长度。
- 变量可存储什么样的信息。
- 可对变量执行什么样的操作。

在传统的语言（如 C 语言）中，类型被内置到语言中。在 C++中，程序员可以通过创建所需的任何类型来扩展该语言，每种新类型都可以行与内置类型相同的功能。

使用结构创建类型的缺点

通过将相关变量组合成结构，提供了在 C 语言中增加新类型的功能。通过使用 typedef 语句，可让结构成为一种新的数据类型。

然而，这种功能有如下缺点：

- 结构和操纵它们的函数不是一个有机的整体：只能通过阅读库的头文件，并使用新类型作为参数进行查找，才能找到函数。
- 对针对结构的相关函数组的行为进行协调非常困难，因为结构中的任何数据可能在任何时候被任何程序逻辑修改。无法防止结构数据不受干扰。
- 内置的运算符不适用于结构：不能使用（+）将两个结构相加，即使这可能是一种最自然的表示问题解决方案的方式（例如，当每个结构表示要组合在一起的一串文本时）。

6.3　类和成员简介

在 C++中，可通过声明一个类来创建一种新类型。类将一组变量（它们的类型通常不同）和一组相关的函数组合在一起。

一种看待汽车的方式是，将其视为车轮、车门、座位、车窗等的集合；另一种方式是考虑其功能：它可以行驶、加速、减速、刹车、停车等。类让你能够将这些部件和功能封装到一个集合中，这种集合被称为对象。

对程序员来说，将有关汽车的一切封装到类中有很多优点。所有一切都在一个地方，这使引用、复制和调用处理数据的函数很容易。同样，类的客户（使用类的程序代码）可以直接使用对象，而不必考虑它包含什么以及它是如何工作的。

类可以由各种类型的变量组成，还可以包含其他类对象。类中的变量被称为成员变量或数据成员。类 Car 可能有表示座位、收音机类型、轮胎等的成员变量。

成员变量也叫数据成员，它们是类中的变量。成员变量是类的组成部分，就像车轮和发动机是汽车的一部分一样。

类还可以包含函数，它们被称为成员函数或方法。成员函数和成员变量一样，也是类的组成部分，它们决定了类的功能。

类中的成员函数通常操纵成员变量。例如，类 Car 可能包含方法 Start( )和 Brake( )。类 Cat 可能包含表示年龄和重量的数据成员，其方法可能包括 Sleep( )、Meow( )和 ChaseMice( )。

6.3.1 声明类

类声明将有关类的信息告诉编译器。要声明类，可使用关键字 class，后跟类名、左大括号、数据成员列表和方法，然后是右大括号和分号。下面是 Cat 类的声明：

```
class Cat
{
    unsigned int itsAge;
    unsigned int itsWeight;
    void Meow();
};
```

上述声明并没有为 Cat 分配内存，它只是告诉编译器 Cat 是什么：它包含哪些数据（itsAge 和 itsWeight），有何功能（Meow()）。虽然没有分配内存，但声明确实让编译器知道 Cat 有多大，即编译器必须为你创建的每个 Cat 对象预留多少内存。在这个例子中，如果 int 变量占 4 字节，则 Cat 的大小为 8 字节：itsAge 占 4 字节，itsWeight 占 4 字节。Meow()只占用存储有关它所处位置的信息所需的空间，这是一个函数指针，在 32 位平台上，占用 4 字节。

6.3.2 有关命名规则的说明

作为程序员，必须给所有成员变量、成员函数和类命名。正如第 3 章指出的，这些名称应易于理解且有意义。Cat、Rectangle、Employee 都是很好的类名。而 Meow()、ChaseMice()和 StopEngine 都是很好的函数名，因为它们指出了函数的功能。很多程序员在成员变量名中使用前缀 its，如 itsAge、itsWeight、itsSpeed。这有助于将成员变量和非成员变量区分开来。

有些程序员使用其他的前缀，如 myAge、myWeight、mySpeed。还有一些人使用字母 m，或再加上下划线，如 mAge 或 m_age、mWeight 或 m_weight、mSpeed 或 m_speed。

有些程序员喜欢在类名称前加一个特殊字符，如 cCat 和 cPerson，而另外一些人喜欢全部大写或小写。本书采用首字母大写的方式命名所有类，如 Cat 和 Person。

同样，很多程序员采用首字母大写的方式表示函数，用首字母小写的方式表示变量。单词之间通常用下划线分隔，如 Chase_Mice，或者将每个单词的首字母大写，如 ChaseMice 和 DrawCircle。

重要的是，应选择一种风格，并在整个程序中始终使用这种风格。随着时间的推移，你的风格会发展到涵盖命名规则、缩进、大括号对齐方式和注释风格。

注意：软件开发公司通常在风格方面有内部标准。这可确保所有开发人员都轻松地读懂其他开发人员编写的代码。不幸的是，这种趋势延伸到了开发操作系统和可重用类库的公司，这通常意味着 C++程序必须处理多种不同的命名规则。

警告：正如以前指出的，C++是区分大小写的，因此所有的类、函数和变量名应遵循相同的模式，这样就无需考虑拼写：是 Rectangle、rectangle 还是 RECTANGLE。

6.3.3 定义对象

声明一个类后，便可以将其用作新类型来声明这种类型的变量。声明新类型对象的方式与声明整型变量相同：

```
unsigned int GrossWeight;     // define an unsigned integer
Cat Frisky;                   // define a Cat
```

上述代码定义了一个名为 GrossWeight 的变量，其类型为 unsigned int；还定义了一个 Cat 类对象 Frisky。

6.3.4 类与对象

你不会将猫的定义视为宠物，而是将具体的猫作为宠物。你能够区分猫的概念与在起居室到处跑的猫。

同样，C++也对作为猫概念的 Cat 类和每个 Cat 对象进行区分。因此，Frisky 是一个类型为 Cat 的对象，就像 GrossWeight 是一个类型为 unsigned int 的变量一样。

对象是类的实例。

6.4　访问类成员

定义实际的 Cat 对象（如 Cat Firsky;）后，便可以使用句点运算符（.）来访问该对象的成员。因此，要将 50 赋给 Firsky 的成员变量 Weight，可以这样编写代码：

```
Frisky.itsWeight = 50;
```

同样，要调用 Meow( )函数，可以这样编写代码：

```
Frisky.Meow();
```

使用类方法时，调用了该方法。在这个例子中，对 Frisky 调用了方法 Meow( )。

6.4.1　给对象而不是类赋值

在 C++中，只能给变量赋值，而不能给类型赋值。例如，绝不要编写这样的代码：

```
int = 5;            // wrong
```

编译器将认为这条语句是错误的，因为不能将 5 赋给整型，而必须定义一个整型变量，然后将 5 赋给该变量。例如：

```
int  x;             // define x to be an int
x = 5;              // set x's value to 5
```

这段代码的意思是：将 5 赋给类型为 int 的变量 x。同样，也不能编写这样的代码：

```
Cat.itsAge=5;       // wrong
```

编译器将视该语句为错误，因为不能将 5 赋给 Cat 类的年龄部分，而必须先定义一个 Cat 对象，然后再将 5 赋给该对象，例如：

```
Cat Frisky;         // just like int x;
Frisky.itsAge = 5;  // just like x = 5;
```

6.4.2　类不能有没有声明的功能

请做这样一个实验：走到一个 3 岁小孩面前，给他看一只猫，然后说：“这是 Frisky，Frisky 会耍把戏。Frisky，汪汪”。这个小孩会格格笑着说：“不，你真笨，猫不会像狗那样叫”。

同样，如果编写如下代码：

```
Cat  Frisky;            // make a Cat named Frisky
Frisky.Bark()           // tell Frisky to bark
```

编译器会说：不，笨蛋，猫不会汪汪叫（编译器的提示语可能是“[531]Error: Member function Bark not found in class Cat.”）。编译器之所以知道 Frisky 不能汪汪叫，是因为 Cat 类没有 Bark( )函数。如果没有定义 Meow( )函数，编译器甚至不会让 Frisky 喵喵叫。

应该：

一定要使用关键字 class 来声明类。

使用句点运算符（.）来访问类的成员和函数。

不应该：

不要把声明与定义混为一谈。声明指出类是什么，定义为对象分配内存。

不要将类与对象混为一谈。

不要将值赋给类，而应将其赋给对象的数据成员。

6.5 私有和公有

在类声明中还常常使用其他关键字，其中最重要的两个是 public 和 private。

关键字 public 和 private 用于类的成员：数据成员和成员方法。私有成员只能在类的方法中访问；公有成员可以通过类的任何对象进行访问。这种区分既重要又令人迷惑。默认情况下，所有类成员都是私有的。

为解释得清楚点，来看本章前面的一个例子：

```
class Cat
{
   unsigned int  itsAge;
   unsigned int  itsWeight;
   void Meow();
};
```

在这个声明中，itsAge、itsWeight 和 Meow( )都是私有的，因为类的所有成员默认均为私有。这意味着除非特别说明，否则它们都是私有的。如果你创建一个程序，并在 main( )中编写如下代码：

```
int main()
{
    Cat Boots;
    Boots.itsAge=5;       // error! Can't access private data!
    ...
```

编译器将认为有错误。通过让这些成员为私有的，你实际上告诉了编译器，我将只在 Cat 类的成员函数中访问 itsAge、itsWeight 和 Meow( )；而在上述代码中，你从 Cat 方法的外部访问 Boots 对象的成员变量 itsAge。仅仅由于 Boots 是 Cat 类的一个对象，并不意味着可以访问 Boots 的私有部分。

C++新手经常会对此感到迷惑。几乎每天都有人在喊：喂！我将 Boots 定义为一个 Cat，为什么 Boots 不能访问自己的年龄？答案是 Boots 能，但你不能。Boots 在自己的方法中可以访问所有部分：私有部分和公有部分。虽然你创建了一个 Cat，但这并不意味着你可以看到或修改它的私有部分。

要能够使用 Cat 的数据成员，可以将有些成员声明为公有的：

```
class Cat
{
  public:
    unsigned int  itsAge;
    unsigned int  itsWeight;
    void Meow();
};
```

在这个声明中，itsAge、itsWeight 和 Meow( )都为公有，这样前面的 Boots.itsAge=5 将能够通过编译，而不会出现任何问题。

注意：关键字 public 将被应用于声明中关键字 private 之前的所有成员，反之亦然。这让你能够轻松地声明类的私有部分和公有部分。

程序清单 6.1 是包含公有成员变量的类 Cat 的声明。

程序清单 6.1 访问一个简单类的公有成员

```
1: // Demonstrates declaration of a class and
```

```
2:  // definition of an object of the class
3:
4:  #include <iostream>
5:
6:  class Cat                // declare the Cat class
7:  {
8:    public:                // members that follow are public
9:      int itsAge;          // member variable
10:     int itsWeight;       // member variable
11: };        // note the semicolon
12:
13: int main()
14: {
15:    Cat Frisky;
16:    Frisky.itsAge = 5;     // assign to the member variable
17:    std::cout << "Frisky is a cat who is ";
18:    std::cout << Frisky.itsAge << " years old.\n";
19:    return 0;
20: }
```

输出：

```
Frisky is a cat who is 5 years old.
```

分析：

第 6 行包含关键字 class，这告诉编译器，接下来是一个声明。新类的名称位于关键字 class 的后面，这里为 Cat。

声明体从第 7 行的左大括号开始，到第 11 行的右大括号和分号结束。第 8 行包含关键字 public 和一个冒号，这表示接下来所有的成员均为公有，直到遇到 private 或到达类声明末尾为止。

第 9 和 10 行包含类成员 itsAge 和 itsWeight 的声明。

从第 13 行开始是程序的 main()函数。第 15 行将 Frisky 定义为一个 Cat 实例，即 Cat 对象。在第 16 行，Frisky 的年龄被设置为 5。第 17 和 18 行使用成员变量 itsAge 打印了一条有关 Frisky 的消息。读者应注意在第 16 和 18 行是如何访问对象 Frisky 的成员的，itAge 是使用对象名（这里为 Frisky）、句点和成员名（这里为 itsAge）访问的。

注意：试着注释掉第 8 行，再重新编译，将收到一条第 16 行有错的消息，因为 itsAge 将不再是公有成员。注释掉这行后，itsAge 和其他成员将默认为私有的。

使数据成员私有

一个通用的设计规则是，应让类的数据成员为私有的。当然，你可能会问，如果将所有数据成员都设置为私有的，如何访问有关类的信息。例如，如果 itsAge 是私有的，如何能够获得或设置 Cat 对象的年龄？

为访问类中的私有数据，必须创建被存取器方法（accessor method）的公有函数。这些方法用于设置和获取私有成员变量，它们是成员函数，可以在程序的其他地方调用它们来获取和设置私有成员变量。

公有存取器方法是类成员函数，用于读取或设置私有类成员变量的值。

为什么要费力地使用这种额外的间接访问呢？在直接使用数据更简单也更容易的情况下，为何要添加这些额外的函数呢？为何要通过存取器函数来工作？

这些问题的答案是，存取器函数让你能够将数据的存储细节与数据的使用细节分开。通过使用存取器函数，以后修改数据的存储方式时，不必重新编写使用这些数据的函数。

如果需要知道 Cat 的年龄的函数直接访问 itsAge，当 Cat 类的作者决定改变该数据的存储方式时，必须重新编写该函数。通过让函数调用 GetAge()，Cat 类可以轻松地返回正确的值，而不管得到年龄的方式如何。

调用函数不需要知道年龄是被存储在 unsigned int 变量或 long 变量中，还是在需要时计算得到的。

这种技术使得程序更容易维护。它延长了代码的生命周期，因为设计的变化不会导致程序作废。

另外，存取器函数还可以包含其他的逻辑。例如，如果 Cat 的年龄不太可能超过 100 岁，或其重量不太可能超过 1 000，在这种情况下，可以禁止这些值。存取器函数可以实施这种限制，还可能完成其他任务。

程序清单 6.2 对 Cat 类进行了修改，使之包含私有成员数据和公有存取器函数。注意，这不是可运行的程序。

程序清单 6.2　包含存取器方法的类

```
1:  // Cat class declaration
2:  // Data members are private, public accessor methods
3:  // mediate setting and getting the values of the private data
4:
4:  class Cat
5:  {
6:    public:
7:      // public accessors
8:      unsigned int GetAge();
9:      void SetAge(unsigned int Age);
10:
11:     unsigned int GetWeight();
12:     void SetWeight(unsigned int Weight);
13:
14:     // public member functions
15:     void Meow();
16:
17:     // private member data
18:   private:
19:     unsigned int  itsAge;
20:     unsigned int  itsWeight;
21: };
```

分析：

这个类有 5 个公有方法。第 8 和 9 行包含 itsAge 的存取器方法，正如读者看到的，第 8 行有一个获取年龄的方法，第 9 行有一个设置年龄的方法。第 11 和 12 行包含 itsWeight 的存取器方法。这些存取器函数设置和返回成员变量的值。

第 15 行声明了公有成员函数 Meow( )，它不是存取器函数：不获取或设置成员变量的值，而是为这个类提供另一项服务——打印单词 Meow。

第 19 和 20 行声明了成员变量。

要设置 Frisky 的年龄，可以将年龄传递给 SetAge( )方法，如下所示：

```
Cat  Frisky;
Frisky.SetAge(5);   // set Frisky's age using the public accessor
```

本章后面将列出 SetAge( )和其他方法的代码。

将方法或数据声明为私有的让编译器能够发现编程错误，避免它们成为 bug。只要愿意支付咨询费，任何程序员都可以获得避开私有性的方法。C++的发明者 Stroustrup 说："C++的访问控制机理用于防范意外事件，而不是保护欺骗"。

关键字 class

关键字 class 的语法如下：

```
class class name
{
```

```
    // access control keywords here
    // class variables and methods declared here
};
```

关键字 class 用于声明新类型。类是类成员数据的集合，类成员数据可以是各种类型的变量，包括其他类。类还包含类函数（方法），即用来操纵类中的数据或为类提供其他服务的函数。

定义类对象的方法与定义变量一样。首先指出类型（类），然后是变量名（对象）。使用句点运算符（.）可访问类的成员和函数。

使用访问控制关键字可将类的某部分声明为私有或公有的。默认的访问控制为私有。访问控制关键字将改变从当前位置到下个关键字或类结尾之间所有成员的访问控制。类声明以有大括号和分号结尾。

范例 1：

```
class Cat
{
   public:
      unsigned int Age;
      unsigned int Weight;
      void Meow();
};

Cat Frisky;
Frisky.Age = 8;
Frisky.Weight = 18;
Frisky.Meow();
```

范例 2：

```
class Car
{
   public:                          // the next five are public

      void Start();
      void Accelerate();
      void Brake();
      void SetYear(int year);
      int GetYear();

   private:                         // the rest is private

      int Year;
      Char Model [255];
};                                  // end of class declaration

Car OldFaithful;                    // make an instance of car
int bought;                         // a local variable of type int
OldFaithful.SetYear(84) ;           // assign 84 to the year
bought = OldFaithful.GetYear();     // set bought to 84
OldFaithful.Start();                // call the start method
```

应该：

务必将存取器方法声明为公有。

必须通过类的成员函数来访问私有成员变量。

不应该：

尽可能不要将成员变量声明为公有的。

不要试图在类的外部使用私有成员变量。

6.6 实现类方法

正如读者看到的，存取器函数提供了到类的私有成员数据的公有接口。每个存取器函数以及声明的其他类方法都必须有实现。实现被称为函数的定义。

成员函数的定义类似于常规函数：首先指出函数的返回类型，如果函数不返回任何值，则使用 void。然后是类名、两个冒号、函数名和参数。程序清单 6.3 给出了简单的 Cat 类的完整声明及存取器函数和一个普通类成员函数的实现。

程序清单 6.3 实现一个简单类的方法

```
1:  // Demonstrates declaration of a class and
2:  // definition of class methods
3:  #include <iostream>            // for cout
4:
5:  class Cat                      // begin declaration of the class
6:  {
7:    public:                      // begin public section
8:      int GetAge();              // accessor function
9:      void SetAge (int age);     // accessor function
10:     void Meow();               // general function
11:   private:                     // begin private section
12:     int itsAge;                // member variable
13: };
14:
15: // GetAge, Public accessor function
16: // returns value of itsAge member
17: int Cat::GetAge()
18: {
19:   return itsAge;
20: }
21:
22: // definition of SetAge, public
23: // accessor function
24: // sets itsAge member
25: void Cat::SetAge(int age)
26: {
27:   // set member variable itsAge to
28:   // value passed in by parameter age
29:   itsAge = age;
30: }
31:
32: // definition of Meow method
33: // returns: void
34: // parameters: None
35: // action: Prints "Meow"to screen
36: void Cat::Meow()
37: {
```

```
38:    std::cout << "Meow.\n";
39: }
40:
41: // create a cat, set its age, have it
42: // meow, tell us its age, then meow again.
43: int main()
44: {
45:    Cat Frisky;
46:    Frisky.SetAge(5);
47:    Frisky.Meow();
48:    std::cout << "Frisky is a cat who is ";
49:    std::cout << Frisky.GetAge() << " years old.\n";
50:    Frisky.Meow();
51:    return 0;
52: }
```

输出：

```
Meow.
Frisky is a cat who is 5 years old.
Meow.
```

分析：

第 5~13 行是 Cat 类的定义。第 7 行的关键字 public 告诉编译器，接下来是一组公有成员。第 8 行声明了公有存取器方法 GetAge( )。GetAge( )让你能够访问第 12 行定义的私有成员变量 itsAge。第 9 行声明了公有存取器函数 SetAge( )。SetAge( )接受一个 int 参数并将 itsAge 设置为该参数的值。

第 10 行声明了类方法 Meow( )。Meow( )不是存取器函数，而是一个将单词 Meow 打印到屏幕上的普通方法。

第 11 行开始为私有部分，其中只包含第 12 行的私有成员变量 itsAge 的声明。第 13 行用右大括号和分号结束类声明。

第 17~20 行包含成员函数 GetAge( )的定义。这个函数没有参数，但返回一个 int 值。从第 17 行可知，类方法包括类名、两个冒号和函数名。这种语法告诉编译器，这里定义的函数 GetAge( )是在 Cat 类中声明的那个。除函数头外，GetAge( )函数的创建方法与任何其他函数相同。

GetAge( )函数只有一行，它返回 itsAge 的值。注意，由于 itsAge 是 Cat 类私有的，因此在 main( )函数中不能直接访问它，但 main( )可以访问公有方法 GetAge( )。

由于 GetAge( )是 Cat 类的一个成员函数，它可以访问变量 itsAge。这使得 GetAge( )能够将 itsAge 的值返回给 main( )。

第 25 行是成员函数 SetAge( )的定义，它接受一个 int 参数(age)，且不返回任何值(用 void 表示)。SetAge( )接受参数 age 的值，将其赋给 itsAge（第 29 行）。由于 SetAge( )函数也是 Cat 类的成员，因此可以直接访问私有成员变量 itsAge。

从第 36 行开始是 Cat 类的 Meow( )方法的定义（实现）。这个函数只有一行，它将单词 Meow 打印到屏幕上并换行。还记得吗，字符\n 表示在屏幕上换行。Meow( )与存取器函数类似，也以返回类型打头，然后是类名、函数名和参数（这个函数没有）。

从第 43 行开始是程序体，也包含函数 main( )。在第 45 行，main( )函数声明了一个名为 Frisky 的类型为 Cat 的对象；也可以这样说，main( )声明一个名为 Frisky 的 Cat。

在第 46 行，通过存取器方法 SetAge( )将 5 赋给成员变量 itsAge。注意，该方法是使用对象名（Frisky）、句点运算符（.）和方法名（SetAge( )）调用的。也可以以类似的方式调用类中的其他方法。

注意：术语成员函数的方法的含义相同。

第47行调用成员函数 Meow()，第48行使用 GetAge()方法打印一条消息。第50行再次调用 Meow()。虽然这些方法都是类（Cat）的一部分，且需要通过对象（Frisky）来调用，但其运行方式与常规函数相同。

6.7 添加构造函数和析构函数

定义整型变量的方法有两种。一种是先定义该变量，然后在程序中为其赋值。例如：

```
int Weight;              // define a variable
...                      // other code here
Weight = 7;              // assign it a value
```

另一种方法是定义该变量并立即对其初始化。例如：

```
int Weight=7;            //define and initialize to 7
```

初始化将变量的定义和赋初值合而为一。以后可以随意修改这个值。初始化确保变量总是有一个有意义的值。

如何初始化类的成员数据呢？可以使用一个特殊的成员函数：构造函数。构造函数可以根据需要接受参数，但它不能有返回值——连 void 都不行。构造函数是一个与类同名的类方法。

声明构造函数后，还应声明析构函数。构造函数创建并初始化类对象，而析构函数在对象被销毁后完成清理工作并释放（在构造函数或对象的生命周期中）分配的资源或内存。析构函数总是与类同名，但在前面加上了一个（~）。析构函数没有参数，也没有返回值。如果要为 Cat 类声明一个析构函数，声明与下面类似：

```
~Cat();
```

6.7.1 默认构造函数和析构函数

有很多种类型的构造函数，有些接受参数，有些不接受。没有参数的构造函数被称为默认构造函数。只有一个析构函数参数：和默认构造函数一样，它也不接受任何参数。

如果没有创建构造函数或析构函数，编译器将为你提供一个。编译器提供的构造函数是默认构造函数。

编译器创建的默认构造函数和析构函数不仅没有参数，而且好像不执行任何操作！如果希望它们执行一些操作，必须创建自己的默认构造函数或析构函数。

6.7.2 使用默认构造函数

什么也不做的构造函数有什么好处呢？在某种程度上说，这是一个格式的问题。所有对象都必须被构造和析构。在构造和析构过程中，将调用这些不执行任何操作的函数。

要在声明对象时不用传递任何参数，如：

```
Cat Rags;          // Rags gets no parameters
```

必须有一个下面这样的构造函数：

```
Cat();
```

创建类对象时，将调用构造函数。如果 Cat 构造函数接受两个参数，应这样创建 Cat 对象：

```
Cat Frisky (5,7);
```

在这个例子中，第一个参数可能是年龄，第二个参数可能是重量。

如果构造函数接受一个参数，应这样编写代码：

```
Cat Frisky (3);
```

如果构造函数根本不接受参数（也就是说，它是默认构造函数），应去掉括号，这样编写代码：

```
Cat Frisky;
```

通常调用函数时，即使函数不接受任何参数也应提供括号，但上面是一种例外情况。这就是可以编写如下代码的原因：

```
Cat Frisky;
```

这行代码被解释为调用默认构造函数。没有提供任何参数，因此将括号删除。

注意，并非一定要使用编译器提供的默认构造函数，可以编写自己的默认构造函数：不接受任何参数的构造函数。也可以给自己的默认构造函数提供函数体，在其中对对象进行初始化。由于格式方面的考虑，建议至少定义一个构造函数，将成员变量指定合适的默认值，以确保对象总是能正确地运行。

同样由于格式方面的考虑，如果声明了一个构造函数，一定要声明一个析构函数，哪怕该析构函数什么也不做。虽然默认析构函数可以正常工作，但声明自己的析构函数并没有坏处。这将让代码更清晰。

程序清单 6.4 重写了 Cat 类，使用一个非默认构造函数来初始化 Cat 对象，将其年龄设置为你提供的初始年龄。该程序清单还演示了在什么地方调用析构函数。

程序清单 6.4　使用构造函数和析构函数

```
// Demonstrates declaration of  constructors and
// destructor for the Cat class
// Programmer created default constructor
#include <iostream>          // for cout

class Cat                    // begin declaration of the class
{
  public:                    // begin public section
    Cat(int initialAge);     // constructor
    ~Cat();                  // destructor
    int GetAge();            // accessor function
    void SetAge(int age);    // accessor function
    void Meow();
  private:                   // begin private section
    int itsAge;              // member variable
};

// constructor of Cat,
Cat::Cat(int initialAge)
{
   itsAge = initialAge;
}

Cat::~Cat()                  // destructor, takes no action
{
}

// GetAge, Public accessor function
// returns value of itsAge member
int Cat::GetAge()
{
   return itsAge;
}

// Definition of SetAge, public
// accessor function
void Cat::SetAge(int age)
```

```
38: {
39:    // set member variable itsAge to
40:    // value passed in by parameter age
41:    itsAge = age;
42: }
43:
44: // definition of Meow method
45: // returns: void
46: // parameters: None
47: // action: Prints "meow"to screen
48: void Cat::Meow()
49: {
50:    std::cout << "Meow.\n";
51: }
52:
53: // create a cat, set its age, have it
54: // meow, tell us its age, then meow again.
55: int main()
56: {
57:    Cat Frisky(5);
58:    Frisky.Meow();
59:    std::cout << "Frisky is a cat who is ";
60:    std::cout << Frisky.GetAge() << " years old.\n";
61:    Frisky.Meow();
62:    Frisky.SetAge(7);
63:    std::cout << "Now Frisky is " ;
64:    std::cout << Frisky.GetAge() << " years old.\n";
65:    return 0;
66: }
```

输出：

```
Meow.
Frisky is a cat who is 5 years old.
Meow.
Now Frisky is 7 years old.
```

分析：

除第 9 行添加一个接受一个 int 参数的构造函数外，程序清单 6.4 与程序清单 6.3 极其相似。第 10 行声明了析构函数，它没有参数。析构函数不能有参数，构造函数和析构函数都没有返回值——一连 void 也没有。

第 19~22 行是构造函数的实现，它与存取器函数 SetAge( )的实现类似。正如读者看到的，构造函数名前加上了类名；前面指出过，这指出方法（这里为 Cat( )）是 Cat 类的一部分。这是一个构造函数，因此没有返回值——一连 void 也没有。然而，该构造函数接受一个初始值，在第 21 行，这个初始值被赋给了数据成员 itsAge。

第 24~26 行是析构函数~Cat( )的实现。该函数不执行任何操作，但如果在类声明中声明了析构函数，必须包含其定义。与构造函数和其他方法一样，在析构函数名前也加上了类名。与构造函数一样（但不同于其他方法），析构函数没有返回类型，也没有参数。这是一个有关析构函数的标准。

第 57 行创建了一个 Cat 对象 Frisky，并将 5 传递给构造函数。从第 60 行可知，不需要调用 SetAge( )函数，因为创建 Frisky 时已经将其成员变量 itsAge 设置为 5。在第 62 行，将 Frisky 的成员变量 itsAge 重新设置为 7。第 64 行打印新的值。

应该：

一定要使用构造函数来初始化对象。

添加构造函数后，一定要添加一个析构函数。

不应该：

不要让构造函数和析构函数有返回值。

不要让析构函数接受任何参数。

6.8　const 成员函数

在本书前面，读者使用关键字 const 来声明不能修改的变量。也可将关键字 const 用于类中的成员函数。如果将类方法声明为 const，必须保证该方法不会修改任何类成员的值。

要将类方法声明为 const，可在方法声明中将所有参数括起的括号和分号之间放置关键字 const，例如：

```
void SomeFunction() const;
```

这声明了一个名为 SomeFunction( )的 const 成员方法，它不接受任何参数，返回类型为 void。由于它被声明为 const 的，因此不会修改其所属类的任何数据成员。

通常使用修饰符 const 将只读取值的存取器函数声明为 const 函数。前面的 Cat 类有两个存取器函数：

```
void SetAge(int anAge);
int GetAge();
```

函数 SetAge( )不能是 const 的，因为它修改成员变量 itsAge 的值；而 GetAge( )应该是 const 的，因为它不修改类的任何成员。

GetAge( )只是返回成员变量 itsAge 的当前值。因此这些函数的声明应该写成这样：

```
void SetAge(int anAge);
int GetAge() const;
```

如果将一个函数声明为 const 的，而该函数的实现通过修改某个成员变量的值而修改了对象，编译器将其视为错误。例如，如果将前面的 GetAge( )声明为 const 的，而记录询问 Cat 年龄的次数，将产生编译错误。这是因为调用该方法将修改 Cat 对象。

一种良好的编程习惯是，尽可能将方法声明为 const 的。这样让编译器捕获错误，而不致于成为等到程序运行时才出现的 bug。

6.9　接口与实现

客户是程序中创建和使用类对象的部分。可以将类的公有接口（类声明）视为与客户的合同。这个合同指出了类的行为方式。

例如，在 Cat 类的声明中，签订了这样一份合同：Cat 的年龄可在构造函数中初始化、使用存取器函数 SetAge( )进行赋值，还可以使用存取器函数 GetAge( )来读取。另外还承诺：Cat 对象都知道如何使用 Meow( )。注意，在公有接口中没有对成员变量 itsAge 作任何说明；这是实现细节，不是合同的一部分。类能够提供年龄（GetAge( )），也能够设置年龄（SetAge( )），但机制（itsAge）是不可见的。

如果将 GetAge( )设置为 const 函数（也应该这样做），合同将承诺：被调用时，GetAge( )不会修改 Cat。

C++是强类型化（strongly typed）语言，这意味着如果你违反条款，编译器将通过显示编译错误强制执行合同。程序清单 6.5 演示了一个因违反合同而不能通过编译的程序。

警告：程序清单 6.5 不能通过编译！

程序清单 6.5　违反接口合同的例子

```
// Demonstrates compiler errors
// This program does not compile!
#include <iostream>          // for cout

class Cat
{
  public:
    Cat(int initialAge);
    ~Cat();
    int GetAge() const;          // const accessor function
    void SetAge (int age);
    void Meow();
  private:
    int itsAge;
};

// constructor of Cat,
Cat::Cat(int initialAge)
{
   itsAge = initialAge;
   std::cout << "Cat Constructor\n";
}

Cat::~Cat()                  // destructor, takes no action
{
   std::cout << "Cat Destructor\n";
}
// GetAge, const function
// but we violate const!
int Cat::GetAge() const
{
   return (itsAge++);        // violates const!
}

// definition of SetAge, public
// accessor function

void Cat::SetAge(int age)
{
   // set member variable its age to
   // value passed in by parameter age
   itsAge = age;
}

// definition of Meow method
// returns: void
// parameters: None
// action: Prints "meow"to screen
void Cat::Meow()
{
```

```
51:    std::cout << "Meow.\n";
52: }
53:
54: // demonstrate various violations of the
55: // interface, and resulting compiler errors
56: int main()
57: {
58:    Cat Frisky;              // doesn't match declaration
59:    Frisky.Meow();
60:    Frisky.Bark();           // No, silly, cat's can't bark.
61:    Frisky.itsAge = 7;       // itsAge is private
62:    return 0;
63: }
```

分析：

这个程序不能编译，因此没有输出。

这个程序有很多错误，编写它只是为了说明问题。

第 10 行将 GetAge( )声明为 const 存取器函数，本应该如此。然而，在 GetAge( )函数体中，第 32 行将成员变量 itsAge 递增。由于该方法被声明为 const 的，因此不能修改 itsAge 的值。因此，编辑该程序时，这将被视为错误。

在第 12 行，Meow( )没有被声明为 const 的。这虽然不是错误，但是一种糟糕的编程习惯。一种更好的设计是，应考虑这个方法不会修改 Cat 的成员变量。因此，Meow( )应被声明为 const 的。

第 58 行创建了一个 Cat 对象 Frisky。Cat 现在有一个构造函数，该构造函数接受一个 int 参数。这意味着必须传递一个参数，而第 58 行并没有传递，因此被视为错误。

注意：只要你提供了构造函数，编译器就不会提供。因此，如果创建了一个接受一个参数的构造函数，除非自己编写默认构造函数，否则将没有默认构造函数。

第 60 行调用类方法 Bark( )，而 Bark( )从未声明，因此，这是非法的。

第 61 行将 7 赋给 itsAge。由于 itsAge 是私有数据成员，因此编译该程序时，这将被视为错误。

为什么使用编译器来捕获错误

编写 100%正确的代码当然很好，但很少有程序员能做到这一点。然而，很多程序员开发了一个系统，通过在编程初期找出并修复错误，来最大限度地减少 bug。

虽然编译器错误令人头痛，对程序员的生存也是一种挑战，但与其他错误相比，这种错误要好得多。对于弱类型语言，当你违反合同时，编译器不会发出错误信号，但是当你的老板正在你旁边观看你运行程序时，程序可能崩溃。更糟糕的是，在捕获错误方面，测试的帮助相对较小，因为对于实际的程序而言，能够通过测试的路径实在太多了。

编译错误（编译时出现的错误）比运行错误（运行程序时出现的错误）要好得多，因为编译错误被查出的可能性很大。即使运行程序多次，也可能没有经过每个可能的代码路径。因此，运行错误可能潜伏很长时间，而编译错误在每次编译时都会出现，因此更容易发现和修复。高质量编程的目标是，确保代码没有运行错误。为实现这种目标，一种经过实践检验的方法是，使用编译器在开发初期捕获错误。

6.10　将类声明和方法定义放在什么地方

为类声明的每个函数都必须有定义，这种定义被称为函数实现。与其他函数一样，类方法的定义也由函

数头和函数体组成。

定义必须位于编译器能够找到的文件中，大多数 C++编译器希望这种文件的扩展名为.c 或.cpp。本书使用.cpp，但请了解你的编译器要求使用哪种扩展名。

注意：很多编译器假定扩展名为.c 的文件为 C 程序，而 C++程序使用扩展名.cpp。你可以使用任何扩展名，但使用.cpp 可最大限度地避免混淆。

也可以将声明放在实现文件中，但这不是一种好的编程习惯。大多数程序员采用的约定是，将声明放在头文件中，该头文件的名称与实现文件相同，但扩展名为.h、.hp 或.hpp。本书将.hpp 用作头文件的扩展名，但请了解你的编译器要求使用何种扩展名。

例如，可以将 Cat 类的声明放在一个名为 Cat.hpp 的文件中，而将类方法的定义放在一个名为 Cat.cpp 的文件中。然后，在 Cat.cpp 的开头加入如下代码，将头文件同.cpp 文件关联起来：

```
#include "Cat.hpp"
```

这段代码告诉编译器，将 Cat.hpp 读入文件中，就好像在这里直接输入了头文件的内容一样。注意，有些编译器要求#include 语句和文件系统中的文件名大小写一致。

如果只想将.hpp 文件的内容读入.cpp 文件中，又何必将它们分成两个文件呢？在大多数情况下，类的客户并不关心实现细节。只要阅读头文件，就可以知道需要知道的所有信息；他们可以忽略实现文件。另外，很可能需要在.cpp 文件中包含同一个.hpp 文件。

注意：类声明告诉编译器：类是什么、它存储什么样的数据、有什么样的函数。之所以将类声明称为接口，是因为它告诉用户如何与类打交道。接口通常存储在扩展名为.hpp 的头文件中。

函数定义告诉编译器，函数是如何工作的。函数定义被称为类方法的实现，存储在一个.cpp 文件中。类的实现细节只有类的作者关心；类的客户（即使用类的那部分程序）不需要知道，也不关心函数是如何实现的。

6.11 内联实现

就像可以请求编译器将常规函数作为内联的一样，也可以将类方法作为内联的，为此只需在返回类型前加上关键字 inline。例如，GetWeight()函数的内联实现如下：

```
inline int Cat::GetWeight()
{
    return itsWeight;    // return the Weight data member
}
```

也可以将函数的定义放到类声明中，这样，函数将自动成为内联的。例如：

```
class Cat
{
   public:
      int GetWeight() { return itsWeight; }  // inline
      void SetWeight(int aWeight);
};
```

请注意 GetWeight()定义的语法。内联函数的函数体紧跟在类方法声明之后，圆括号后面没有分号。与其他函数一样，该定义以左大括号开始，以右大括号结束。和往常一样，空白无关紧要；可以这样书写上述声明：

```
class Cat
{
```

```
public:
    int GetWeight() const
    {
        return itsWeight;
    }                    // inline
    void SetWeight(int aWeight);
};
```

程序清单 6.6 和 6.7 重新编写了 Cat 类，将类声明放在文件 Cat.hpp 中，将函数的实现放在文件 Cat.cpp 中。程序清单 6.7 还将存取器函数和 Meow( )函数改为内联的。

程序清单 6.6　位于 Cat.hpp 中的 Cat 类声明

```
1:  #include <iostream>
2:  class Cat
3:  {
4:    public:
5:      Cat (int initialAge);
6:      ~Cat();
7:      int GetAge() const { return itsAge;}      // inline!
8:      void SetAge (int age) { itsAge = age;}    // inline!
9:      void Meow() const { std::cout << "Meow.\n";}  // inline!
10:   private:
11:     int itsAge;
12: };
```

程序清单 6.7　位于 Cat.cpp 中的 Cat 类实现

```
1:  // Demonstrates inline functions
2:  // and inclusion of header files
3:  // be sure to include the header files!
4:  #include "Cat.hpp"
5:
6:
7:  Cat::Cat(int initialAge)   //constructor
8:  {
9:    itsAge = initialAge;
10: }
11:
12: Cat::~Cat()              //destructor, takes no action
13: {
14: }
15:
16: // Create a cat, set its age, have it
17: // meow, tell us its age, then meow again.
18: int main()
19: {
20:   Cat Frisky(5);
21:   Frisky.Meow();
22:   std::cout << "Frisky is a cat who is " ;
23:   std::cout << Frisky.GetAge() << " years old.\n";
24:   Frisky.Meow();
25:   Frisky.SetAge(7);
26:   std::cout << "Now Frisky is " ;
```

```
27:    std::cout << Frisky.GetAge() << " years old.\n";
28:    return 0;
29: }
```

输出：

```
Meow.
Frisky is a cat who is 5 years old.
Meow.
Now Frisky is 7 years old.
```

分析：

程序清单 6.6 和 6.7 中的代码与程序清单 6.4 中的代码相似，只是将声明放到了 Cat.hpp（程序清单 6.6）中，同时在声明中将 3 个方法声明为内联的。

Cat.hpp 的第 6 行声明了 GetAge()，并提供了其内联实现。第 7 和 8 行给出了另外两个内联函数。但这些内联函数功能与前一个例子中相同。

程序清单 6.7 的第 4 行为#include “Cat.hpp”，它导入 Cat.hpp 的内容。通过包含 Cat.hpp，告诉预编译器将 Cat.hpp 读入到文件中，就像这里（第 5 行）输入了该文件的内容一样。

这种技术让你能够将声明和实现放在不同的文件中，同时让编译器需要时可以使用声明。这是 C++编程中一项非常常见技术。通常，将类声明放在一个.hpp 文件中，然后使用#include 语句将其包含到相关的.cpp 文件中。

第 18～29 行与程序清单 6.4 中的 main()函数相同，用于证明将这些函数声明为内联的并不会改变其功能。

6.12 将他类用作成员数据的类

种常见的创建复杂类的方法是，首先声明较简单的类，然后将其包含到较复杂类的声明中。例如，你可能声明车轮类、发动机类、离合器类等，然后将它们组合成汽车类。这声明一种 has-a（有一个）关系。汽车有发动机、车轮、离合器等。

来看另一个例子。矩形由线段组成，线段由两点确定，而点由坐标 x、y 确定。程序清单 6.8 给出了 Rectangle 类的完整声明，该声明可能存储在文件 Rectangle.hpp 中。由于矩形被定义为连接 4 个点的 4 条线段，而每个点对应图上一个坐标，因此先声明一个 Point 类来存储点的 x、y 坐标。程序清单 6.9 给出了这两个类的实现。

程序清单 6.8 声明一个完整的类

```
1: // Begin Rectangle.hpp
2: #include <iostream>
3: class Point     // holds x,y coordinates
4: {
5:   // no constructor, use default
6:   public:
7:     void SetX(int x) { itsX = x; }
8:     void SetY(int y) { itsY = y; }
9:     int GetX()const { return itsX;}
10:    int GetY()const { return itsY;}
11:  private:
12:    int itsX;
13:    int itsY;
14: };   // end of Point class declaration
15:
16:
```

```
17: class Rectangle
18: {
19:   public:
20:     Rectangle (int top, int left, int bottom, int right);
21:     ~Rectangle () {}
22:
23:     int GetTop() const { return itsTop; }
24:     int GetLeft() const { return itsLeft; }
25:     int GetBottom() const { return itsBottom; }
26:     int GetRight() const { return itsRight; }
27:
28:     Point  GetUpperLeft() const { return itsUpperLeft; }
29:     Point  GetLowerLeft() const { return itsLowerLeft; }
30:     Point  GetUpperRight() const { return itsUpperRight; }
31:     Point  GetLowerRight() const { return itsLowerRight; }
32:
33:     void SetUpperLeft(Point Location)  {itsUpperLeft = Location;}
34:     void SetLowerLeft(Point Location)  {itsLowerLeft = Location;}
35:     void SetUpperRight(Point Location)  {itsUpperRight = Location;}
36:     void SetLowerRight(Point Location)  {itsLowerRight = Location;}
37:
38:     void SetTop(int top) { itsTop = top; }
39:     void SetLeft (int left) { itsLeft = left; }
40:     void SetBottom (int bottom) { itsBottom = bottom; }
41:     void SetRight (int right) { itsRight = right; }
42:
43:     int GetArea() const;
44:
45:   private:
46:     Point itsUpperLeft;
47:     Point itsUpperRight;
48:     Point itsLowerLeft;
49:     Point itsLowerRight;
50:     int   itsTop;
51:     int   itsLeft;
52:     int   itsBottom;
53:     int   itsRight;
54: };
55: // end Rectangle.hpp
```

程序清单 6.9　RECT.cpp

```
1: // Begin Rect.cpp
2: #include "Rectangle.hpp"
3: Rectangle::Rectangle(int top, int left, int bottom, int right)
4: {
5:    itsTop = top;
6:    itsLeft = left;
7:    itsBottom = bottom;
8:    itsRight = right;
9:
10:   itsUpperLeft.SetX(left);
11:   itsUpperLeft.SetY(top);
```

```
12:
13:    itsUpperRight.SetX(right);
14:    itsUpperRight.SetY(top);
15:
16:    itsLowerLeft.SetX(left);
17:    itsLowerLeft.SetY(bottom);
18:
19:    itsLowerRight.SetX(right);
20:    itsLowerRight.SetY(bottom);
21: }
22:
23:
24: // compute area of the rectangle by finding sides,
25: // establish width and height and then multiply
26: int Rectangle::GetArea() const
27: {
28:    int Width = itsRight-itsLeft;
29:    int Height = itsTop - itsBottom;
30:    return (Width * Height);
31: }
32:
33: int main()
34: {
35:    // initialize a local Rectangle variable
36:    Rectangle MyRectangle (100, 20, 50, 80 );
37:
38:    int Area = MyRectangle.GetArea();
39:
40:    std::cout << "Area: " << Area << "\n";
41:    std::cout << "Upper Left X Coordinate: ";
42:    std::cout << MyRectangle.GetUpperLeft().GetX();
43:    return 0;
44: }
```

输出：

```
Area: 3000
Upper Left X Coordinate: 20
```

分析：

在文件 Rectangle.hpp（程序清单 6.8）中，第 3～14 行声明了 Point 类，用来存储 x、y 坐标。这个程序对 Points 用得不多，但其他绘图方法需要 Point。

注意：如果声明一个名为 Rectangle 的类，有些编译器会报告错误。这通常是由于有一个名为 Rectangle 的内部类。如果出现这种问题，只需将类名改为 myRectangle 即可。

在 Point 类的声明中，第 12～13 行声明了两个成员变量（itsX 和 itsY）。这两个变量用于存储坐标值。x 坐标增大时，点向右移动；y 坐标增大时，点向上移动。有些图形使用不同的坐标系，例如，在有些 Windows 程序中，在窗口中向下移动时 y 坐标增加。

Point 类使用第 7～10 行声明的内联存取器函数来获取和设置 x 和 y 的值。Point 类使用默认构造函数和析构函数，因此必须显式地设置点的坐标。

从第 17 行开始是 Rectangle 类的声明。Rectangle 由表示顶角的 4 个点组成。

第 20 行是 Rectangle 的构造函数，它接受 4 个 int 参数，分别是 top、left、bottom 和 right。在程序清单 6.9 中，这 4 个参数的值被复制到 4 个成员变量中，然后创建 4 个点 Points。

除了几个普通的存取器函数外，Rectangle 还有一个 GetArea( )函数，它是在第 43 行声明的。在程序清单 6.9 的第 28 和 29 行，该函数计算矩形的面积，而不是将面积存储在一个变量中。为此，它计算矩形的长和宽，然后将它们相乘。

要得到矩形左上角的 x 坐标，需要访问点 UpperLeft，请求它提供 x 的值。由于 GetUpperLeft( )是 Rectangle 的一个方法，因此可以直接访问 Rectangle 的私有数据，包括 itsUpperLeft。由于 itsUpperLeft 是一个 Point，而 Point 的 itsX 值是私有的，因此 GetUpperLeft( )不能直接访问 itsX，而必须使用公有存取器函数 GetX( )来获取这个值。

实际程序的主体从程序清单 6.9 的第 33 行开始。在第 36 行之前，还没有分配任何内存，因此实际上什么也没有发生。到目前为止，只是告诉了编译器如何创建点和矩形。

在第 36 行，通过给 top、left、bottom 和 right 传递值，创建了一个 Rectangle 对象。

第 38 行创建了一个 int 类型的局部变量 Area。该变量用于存储刚才创建的矩形的面积。该变量被初始化为 Rectangle 的 GetArea( )函数的返回值。Rectangle 类的客户不用了解 GetArea( )的实现就能够创建一个 Rectangle 对象并获取的面积。

Rectangle.hpp 如程序清单 6.8 所示。只需阅读这个头文件（它包含了 Rectangle 类的声明），程序员便知道 GetArea( )返回一个 int。Rectangle 类的用户不关心 GetArea( )是如何完成其工作的。实际上，Rectangle 的作者可以修改 GetArea( )函数，而不会影响使用 Rectangle 类的程序，只要该函数仍返回一个 int。

程序清单 6.9 中的第 42 行看起来有些奇怪，但只要考虑发生的情况，便会很清晰。这行代码获取矩形左上角的 x 坐标，它调用了 Rectangle 类的 GetUpperLeft( )的方法，该方法返回一个 Point，而你想通过该 Point 获取其 x 坐标。你知道在 Point 类中，获取 x 坐标的存取器函数为 GetxX( )。第 42 行只是将存取器函数 GetUpperLeft( )和 GetX( )组合到了一起：

```
MyRectangle.GetUpperLeft().GetX();
```

上述代码获取对象 MyRectangle 的左上角的 x 坐标。

FAQ

声明与定义的区别是什么？

答：声明只引入一个名称而不为其分配内存；而定义分配内存。

存在一些例外情况，在这些情况下，所有声明也是定义。其中最重要的例外是，全局函数的声明（原型）和类的声明（通常是在头文件中）。

6.13　结　　构

与关键字 class 及其相似的一个关键字是 struct，它用来声明结构。在 C++中，结构与类相同，只是其成员默认为公有的。可以像声明类一样声明结构，并给它声明数据成员和函数。事实上，如果遵循显式地声明类的公有和私有部分这种良好的编程习惯，那么结构和类的声明方法没有任何不同。

请将程序清单 6.8 做如下修改：

- 在第 3 行，将 class Point 改为 struct Point。
- 在第 17 行，将 class Rectangle 改为 struct Rectangle。

现在再次运行程序，并比较输出，你将发现没有任何变化。

读者可能会问，为什么这两个关键字的功能相同呢？这是一个历史遗留问题。C++是作为 C 语言的扩展开发的。C 语言有结构，虽然 C 语言中的结构没有类方法。C++的创始人 Bjarne Stroustrup 对结构进行了扩展，为表明这是一种新的扩展功能，将名称改为类，同时修改了成员的默认可见性。这也使得可以在 C++程序中

使用庞大的 C 函数库。

应该：

将类声明放在一个.hpp（头）文件中，将成员函数的实现放在一个.cpp 文件中。

尽可能使用 const。

不应该：

理解类之前不要继续学习后面的内容。

6.14 小　　结

本章介绍了如何使用类创建新的数据类型；还介绍了如何定义这些新类型的变量：对象。

类可以有数据成员，数据成员是各种类型（包括其他类）的变量。类还可以包含成员函数，也叫方法。可以使用成员函数来操纵成员数据和提供其他服务。

类成员（包括数据和函数）可以是私有或公有的。程序的任何部分均可访问公有成员。私有成员只能由类的成员函数访问。类成员默认为私有的。

一种良好的编程习惯是，将类的接口（声明）放在一个头文件中（头文件的扩展名通常为.hpp），然后在代码文件（.cpp）中使用 include 语句来使用它。类方法的实现通常存放在扩展名为.cpp 的文件中。

类构造函数可用于初始化对象的数据成员。类析构函数在对象被销毁时执行，它通常用于释放由类方法分配的内存和其他资源。

6.15 问 与 答

问：类对象的大小如何确定？

答：类对象在内存中的大小由类成员变量的大小的总和决定。类方法只占用少量的内存，这些内存用于存储有关方法位置的信息（指针）。

有些编译器在内存中将变量与某种边界对齐，因此两字节的变量实际占用的内存可能不止两字节。请查看编译器手册，看看是否如此，但就现在而言，读者不用考虑这些细节。

问：如果声明一个包含私有成员 itsAge 的 Cat 类，然后再定义两个 Cat 对象 Frisky 和 Boots，Boots 能访问 Frisky 的 itsAge 成员变量吗？

答：不能，同一个类的不同实例不能彼此访问对方的非公有成员。换句话说，如果 Frisky 和 Boots 都是 Cat 的实例，那么 Frisky 的成员函数可以访问 Frisky 的数据，但不能访问 Boots 的数据。

问：为什么不应将所有成员数据都声明为公有的？

答：将数据成员声明为私有的，让类的客户能够使用数据而不用关心数据如何存储或计算的。例如，假设 Cat 类有一个 GetAge( ) 方法，Cat 类的客户可以询问猫的年龄，而不用知道或关心猫是将其年龄保存在一个成员变量中还是动态计算得到的。这意味着编写 Cat 类的程序员可以在以后修改 Cat 类的设计，而不会要求所有的 Cat 用户都修改其程序。

问：既然在 const 函数对类进行修改会导致编译错误，为什么不能省掉 const 以避免错误呢？

答：如果成员函数在逻辑上不应修改类，则通过使用关键字 const，可让编译器帮助发现一些错误。例如，GetAge( )函数可能没有理由去修改 Cat 类，但实现中可能有下面这行代码：

```
if (itsAge = 100) cout << "Hey! You're 100 years old\n";
```

如果将 GetAge( )声明为 const 的，将导致这段代码被视为错误。由于你的本意是判断 itsAge 是否等于 100，但不小心将 100 赋给了 itsAge。由于这种赋值修改了类，而声明中又指出该方法不会修改类，因此编译器能

够发现这种错误。

这种错误仅仅通过浏览代码很难发现。人们常常只能看到预期的东西。更重要的是，程序可能显得一切正常，但 itsAge 却被设置为一个错误的值，这迟早会导致问题。

问：在 C++程序中有什么理由使用结构吗？

答：很多 C++程序员使用 struct 来声明无任何函数的类。这是沿袭旧的 C 结构的做法。在旧的 C 结构中不能有函数。坦率地讲，这是一种令人迷惑的不良编程习惯。今天无方法结构明天可能需要方法。如果那样的话，要么将其改为类，要么打破你的规则，在结构中添加方法。仅当要调用需要特定结构的老式 C 函数时，才应使用结构。

问：有些从事面向对象编程的人使用术语实例化，这是什么意思？

答：实例化是一种奇怪的说法，指的是使用类创建一个对象。类型为某个类的对象是这个类的一个实例。

6.16　作　　业

作业包括测验和练习，前者帮助加深读者对所学知识的理解，后者提供了使用新学知识的机会。请尽量先完成测验和练习题，然后再对照附录 D 中的答案，继续学习下一章之前，请务必弄懂这些答案。

6.16.1　测验

1. 什么是句点运算符，它的作用是什么？
2. 声明和定义哪个会分配内存？
3. 类声明是类的接口还是实现？
4. 公有数据成员和私有数据成员之间有何区别？
5. 成员函数可以是私有的吗？
6. 成员数据可以是公有的吗？
7. 如果创建两个 Cat 对象，它们的 itsAge 成员数据可以有不同的值吗？
8. 类声明是以分号结尾吗？类方法定义呢？
9. 在 Cat 类中，不接受任何参数且返回类型为 void 的 Meow( )的函数头是什么样的？
10. 调用什么函数去初始化对象？

6.16.2　练习

1. 声明一个名为 Employee 的类，它包含如下数据成员：itsAge、itsyearsOfService 和 itsSalary。
2. 重写 Employee 类的声明，使其数据成员为私有，并提供获得和设置每个数据成员的公有存取器方法。
3. 编写一个程序，使用 Employee 类来创建两个 Employees 对象，设置它们的 itsAge、itsYearOfService 和 itsSalary，并打印它们的值。你还需要添加存取器方法的代码。
4. 继续练习 3，给 Employee 类编写一个方法，它报告雇员挣多少千美元（四舍五入到最接近的千数）。
5. 修改 Employee 类，以便能够在创建 Employee 对象时初始化 itsAge、itsYearOfService 和 itsSalary。
6. 查错：下面的声明有什么错误？

```
class Square
{
   public:
      int Side;
}
```

7. 查错：为什么下面的类声明不是特别有用？

```
class Cat
{
```

```
        int GetAge() const;
    private:
        int itsAge;
};
```

8. 查错：编译器将在下面的代码中找到哪 3 处错误？

```
class  TV
{
    public:
        void SetStation(int Station);
        int GetStation() const;
    private:
        int itsStation;
};

int main()
{
        TV myTV;
        myTV.itsStation = 9;
        TV.SetStation(10);
        TV myOtherTv(2);
}
```