[C++:A Beginner's Guide, Second Edition]

C++初学者指南

[MSDN 初学者开发人员学习中心 指定教材]

编著:Herb Schildt
翻译:zhangxingping@CSDN.NET
整理:Roamr@doPure

第 1 章：C++基础知识

如果说有一种语言定义了当今编程的实质，那么它就是 C++。它是高性能软件开发的杰出语言。它的语法已经成为专业编程语言的标准，并且它的设计思想在计算界引起深刻的反响。

Java 和 C#语言都是从 C++继承而来的。简而言之 ,要成为专业的编程人员就意味着要深刻理解 C++。它是现代编程的基础。

本篇旨在介绍 C++，包括它的历史，它的设计思想，以及几个最重要的特性。学习一门编程语言最困难的事情就是所有的元素都不是单独孤立存在的。相反，构成语言的各个部分是相互协作，一起工作的。这种相关性使得我们很难讨论 C++的一个方面而不去考虑其它方面。为了克服这个困难 ,本篇对几个 C++特性进行了简单的介绍，包括 C++程序的通用形式，一些基本的控制语句，以及运算符。本篇不会涉及过多的细节，更注重 C++程序中通用的概念。

必备技能 1.1：C++历史简介

C++是从 C 语言发展而来的。这一点不难理解 ,因为 C++是构筑的 C 语言的基础之上的。然而 ,C++是 C 语言的超集。C++扩展并增强了 C 语言 ,支持面向对象的编程(这点在本篇的后面会进行描述)。C++同时对 C 语言做了改进，包括扩展了例行程序库集。然而大部分的 C++特性是直接从 C 继承而来的。因此，为了充分理解和欣赏 C++，我们必须深入了解 C 语言是如何运作的。

C：现代编程时代的开始

C 语言的发明定义了现代编程时代的开始。它的影响不应该被低估，因为它从根本上改变了人们考虑和实现程序的方法。它的设计思想和语法已经影响到了每一个主流的编程语言。C 语言是计算发展的主要的，革命性的推动力之一。

C 语言由 Dennis Ritchie 在 DECPDP-11 电脑上，在 UNIX 操作系统下发明并实现的。C 语言是从一种古老的叫做 BCPL 的语言发展而来。BCPL 是由 Martin Richards 开发的。BCPL 语言对由 Ken Thompson 发明的 B 语言产生着深刻的影响，而 B 语言最终在 20 世纪 70 年代发展成 C 语言。

在 C 语言发明之前 ,计算机语言被设计出来或者是为了进行学术计算 ,或者是为官方的委员们所使用。而 C 却不同。它是一群真实的程序员设计、实现并开发出来的。它反映了这些人完成编程工作的方法。它的特性是这群实际使用这个语言的人们反复推敲，打磨，测试的结果。因此，C 语言吸引了众多的拥护者，并迅速成为全世界编程人员的选择。

C 语言的发展经历了 20 世纪 60 年代的结构化编程的革命。在此之前，大型程序是难以书写的，因为程序的逻辑趋向于退化成“意大利面条式的代码”，充斥着难以理解的，混乱的转跳，函数调用和返回。结构化编程通过增加很好的控制语句，带有局部变量的子程序段和其它的一些改进解决了这个问题。结构化编程使得编写巨大的程序成为了可能。尽管还有别的结构化的编程语言，例如 Pascal，C 却是第一个功能强大，富于表达，能书写出优美代码的结构化语言。它语法简单易用，并且它的设计思想是程序员掌控一切，而不是语言本身掌控一切，这就使得 C 语言很快拥有了众多的拥护者。我们现在来看这点可能有点难以理解，但是 C 当时确实为编程者带来了他们渴望已久的新鲜空气。因此，C 语言很快就在 20 世纪 80 年代变成了使用最广泛的结构化编程语言。

我们需要 C++

经过前文的描述，你可能会问，那为什么还要发明 C++呢？既然 C 是很优秀的编程语言，我们为什么还需要别的编程语言呢？这个问题的答案非常复杂。纵观编程技术的发展历史，程序复杂度的增加驱使我们需要更好的方式来管理我们的程序。C++就应运而生了。为了更好地理解增长的程序复杂度与计算机编程语言发展之间的关系，我们需要简单回顾一下计算机编程技术发展的历史。当计算机刚被发明出来的时候，人们使用计算机前面的面板，通过拨动开关来发送二进制的机器指令。这种方式在计算机程序只有几百行的时候还可以工作。随着计算机程序的增大，人们发明了汇编语言，通过使用符号代替机器指令，以便程序员可以处理更大的，更复杂的程序。第一个被广泛使用的计算机语言是 FORTRAN。FORTRAN 语言在起初给人的印象是非常深刻的，当时几乎没有语言能实现编写整洁，易于理解的程序。20 世纪 60 年代，结构化编程诞生了，这正是诸如 C 语言一样的语言所鼓励的编程方法。通过结构化的编程方法，很轻松的编写大型程序第一次成为了可能。然而，即使是使用结构化的编程方法，一旦一个项目到达了一定的规模，其复杂度也就超过了程序员所能管理的范围。在 20 世纪 70 年代，很多项目几乎都处于这种境地。为了解决这种问题，出现了一种新的编程方法：面向对象编程。通过使用面向对象编程，程序员可以处理更大的，更复杂的程序。而 C 语言是不支持这种面向对象编程方法的。于是，人们对面向对象的 C 的渴望就直接导致了 C++的诞生。可见，自从计算机发明以来，编程的方式已经发生了巨大的变化。

最后一点，尽管 C 是世界上最受欢迎的专业编程语言之一，也有复杂的程序是 C 不能完成的。一旦一个程序的规模达到了一定的大小，其复杂度就会增加，以至于很难从整体上对其进行把握。C++的目的就是突破这种障碍，帮助编程人员理解并管理更大，更复杂的程序。

C++的诞生

C++由 Bjarne Stroustrup 于 1979 年在位于新泽西州 Murray Hill 的贝尔实验室成功发明。起初它的名字叫“带有类的 C”，后来在 1983 年更名为 C++。Stroustrup 在 C 的基础上构建了 C++，因此 C++包括 C 的所有特性和优点。它还继承了 C 的理念 程序员而不是程序掌控一切。有一点必须明确，Stroustrup 并没有创建一个全新的编程语言。相反，它增强了已经高度成功的语言。大多数 Stroustrup 为 C 增加的特性都是为了支持面向对象的编程。从本质上来讲，C++就是支持面向对象的 C。通过在 C 的基础上构建 C++，就实现了到面向对象编程的平滑过渡。C 程序员不必重新学习一门新的语言，只需要学习那些新增的特性，就能收获面向对象编程带来的好处。在设计 C++语言的时候，Stroustrup 清楚地知道在增加支持面向对象编程的特性时，保持原有 C 的特性，包括 C 的高效，灵活和 C 的设计理念是非常重要的。幸运的是，他的目标实现了。C++在提供了面向对象的编程优点同时，还保留了 C 的灵活。尽管发明 C++的初衷是为了辅助管理那些大型的程序，但它绝不仅限于此。实际上，C++的面向对象特性可以被有效地引用到实际上任何程序中。C++可以广泛地被用来开发诸如编辑器，数据库，个人文件系统，网络工具，通信程序等，这些都非常常见。由于 C++保留了 C 的高效性，大量的高性能系统软件都是用 C++开发的。同样，C++也经常被用来开发 windows 程序。

C++的发展

C++被发明后，经过了三次大的修订，每次修订都对语言自身做了增加和改动。第一次和第二修订分

别是在 1985 年和 1990 年。第三次修订发生在 C++标准化的过程中。几年前（现在来看，应该是十几年前了），人们开始进行 C++的标准化工作。那时，建立了由美国国家标准研究所（ANSI)和国际标准组织（ISO）合作的标准化组织组。建议标准的第一次草案是在 1994 年 1 月 25 日完成的。在这份草案中，ANSI/ISO 联合委员会保留了 Stroustrup 当初定义的特性，并增加了一些新的特性。总的来说，这份最初的草案反映了当时 C++的情况。在此之后不久，发生了一件事情，促使了联合委员会大大地扩展了该标准：由 Alexander Stepanov 提出的标准模板库的创建。标准模板库是一套我们可以用之处理数据的通用程序的集合。通用模板库很强大，也很简洁优雅。但是它很巨大。在第一次草案之后，委员会曾经投票来决议是否在标准 C++中增加标准模板库。标准模板库的增加使得 C++大大超出了起初定义的范围。对标准模板库和其它一些特性的增加使得 C++标准化的步伐减慢了许多。完全可以说 C++的标准化工作比人们期望的时间要长许多。在整个过程中，C++中加入了许多新的特性，并做了许多小的改动。实际上，由该联合委员会制定的 C++比 Stroustrup 当初设计的 C++要复杂很多。最终的草案在 1997 年 12 月 14 日通过，ANSI/ISO 标准 C++在 1998 年成为现实。这就是人们通常说的标准 C++。本书描述的都是标准的 C++。本书描述的 C++是所有主流 C++编译器，包括微软的 visual C++都支持的 C++。因此本书中的代码和信息是完全可移植的。

必备技能 1.2：C++与 Java 和 C#的关系

除了 C++之外，还有两个重要的现代编程语言：Java 和 C#。Java 是有 Sun Microsystems 公司开发的，而 C#则是由微软公司开发的。由于人们有时会对 C++与 Java 和 C#的关系产生一些混淆，这里有必要对此介绍一下。C++是 Java 和 C#之父。尽管 Java 和 C#都是在 C++的基础上对语言的特性进行了一些增加，删除和改动，但是总体上来说它们三者的语法是几乎相同的。进一步来说，C++所采用的对象模型和 Java，C#的都是相似的。最后，三者给人的总体感觉也是非常相近的。这就意味着，一旦学会了 C++，就能很轻易地学习 Java 和 C#。反之亦然，如果你懂 Java 或者 C#学习 C++也是很简单的。这就是为什么 Java，C#和 C++都是用相同的语法和对象模型了，这也是大量有经验的 C++程序员能顺地过渡到是 Java 或者 C#的原因。它们之间的区别在于各自设计针对的计算环境不同。C++是针对指定类型的 CPU 和操作系统而设计的高性能的语言。例如：如果你想写在 windows 操作系统下，因特尔奔腾系列的 CPU 上运行的程序，那么 C++是最好的选择。

专家答疑

问：

Java 和 C#都实现了跨平台和可移植的编程，C++为什么不能了？

答：

Java 和 C#之所以能实现跨平台，可移植的编程，而 C++不能是因为它们的编译器生成的目标代码不同。就 C++而言，编译器的输出是机器代码，这是 CPU 可以直接执行的。因此它是紧密和指定的 CPU 以及操作系统相关的。如果想让 C++程序在不同的系统上运行，则需要针对该目标系统进行代码的重新编译。为了让 C++程序可以在不同的环境上运行，就需要生成不同的可执行版本。Java 和 C#是通过把代码编译成伪码，一种中间语言。就 Java 而言，这种伪码是在运行时系统上运行的，这就是 Java 虚拟机。对 C#而言，这就是 CLR（公共语言运行时）。因此，Java 语言的程序可以在任何有 java 虚拟机的环境下运行，C#的程序可以在任何实现了 CLR 的环境下运行。因为 Java 和 C#的运行时系统处于程序和 CPU 之间，和 C++相比，这就引起了多余的开销。这就是为什么，对等情况下，C++程序比 Java 和 C#程序运行快的原因了。Java 和 C#的开发是为了满足互联网上在线程序的统一编程需求。（C#的设计也是用来简化软

件构件的开发 ）。互联网上连接的是许多不同的 CPU 和操作系统。因此跨平台和可移植性就成了最重要的着眼点。第一个着眼于这个问题的语言就是 Java。Java 语言编写的程序可以在很多不同的环境下运行。因此 ,Java 程序可以在互联网上自由运行。然而这样做的代价就是牺牲了效率 ,Java 程序的执行要比 C++程序慢许多。同样的事情也发生在 C#身上。最终分析，如果你想开发高性能软件，就是用 C++。如果你想开发高度可移植的软件，就是用 Java 或者 C#。最后一点：请记住，C++，java 和 C#是用来解决不同问题的。这里并没有那个语言好，那个语言不好的问题，而是那个语言更适合用来完成我们手头工作的问题。

练习：

1.C++语言是从什么语言发展而来的？

2.C++语言产生的主要原因是什么？

3.C++语言是 Java 和 C#语言之父，对吗？

答案：

1.C++是从 C 语言发展而来的。

2.程序复杂性的不断增加是产生 C++语言的主要原因。

3.正确。

必备技能 1.3：面向对象的编程

C++的中心是围绕着面向对象的编程。正如前面介绍的那样，面向对象的编程是促进 C++产生的主要因素。正是因为这一点，在学习编写简单的 C++程序之前，理解面向对象编程的基本思想是非常重要的。

面向对象的编程很好地利用了结构化编程的思想，并增加了一些新的概念，能够更好地组织程序。通常情况下，有两种方式来组织程序：以代码为中心或者以数据为中心。通过结构化的编程技术，程序通常是以代码为中心来组织的。这种方法可以被认为是“代码作用于数据”。

面向对象的编程则是以数据为中心。程序以数据为中心进行组织，主要的原则就是“数据控制代码”。在面向对象的语言中，我们定义数据和允许作用于这些数据的例行程序段。因此，数据的类型明确定义了可以作用在这些数据上的操作。

为了支持面向对象编程的原则，所有的面向对象语言，包括 C++，都有三个显著的特点：封装，多态和继承。让我们一个一个来学习。

封装

封装是一种编程机制，它把数据和处理数据的代码捆绑在一起，这可以防止外部程序错误地访问数据和代码。在面向对象的语言中，数据和代码可以通过黑盒子的方式绑定起来，盒子内部是全部必要的数据和代码。当数据和代码以这种方式绑定的时候，此时我们就创建了一个对象。换句话说，一个对象就是支持封装的设备。

专家答疑

问：

我听说过子程序段的学名为方法，那么方法和函数是同一回事情吗？

答：

通常来说，是的。方法在 Java 里面使用很广泛。C++则中叫做函数，Java 中则被称作方法。C#程序员同样使用方法这个术语。因为它已经广泛地被采用了，所以有时候在 C++里面，函数有时候也被称作方法。

对于一个对象来说，代码或者数据或者两者都有可能是私有的或者公有的。私有的代码或者数据只能是该对象的其它部分可以访问和感知的。也就是说，私有的代码和或者数据是不能被对象之外的程序段访问的。当代码或者数据是公有的，虽然它是被定义在对象中的，但是程序的其它部分都是可以访问它的。通常情况下，对象的公有部分是用来提供对对象私有元素的可控的访问接口的。

C++中最基本的封装单元就是类。一个类定义了对象的通用形式。它同时定义了数据和作用于这些数据的代码。C++使用类来构建对象。对象是类的实例。因此，一个类实际上是构建对象的工厂。

类中的数据和代码称为类的成员。更详细一点来说，成员变量，也叫做实例变量，是类定义的数据。成员函数是作用于这些数据的操作。函数在 C++中是针对子程序的术语。

多态

多态，字面意思就是多种形态的意思。它是一种机制，可以允许一个接口来访问类的通用动作。一个简单的多态性的例子就是汽车的方向盘。方向盘（接口）都是一样的，不管实际中使用了怎样的转向机制。也就是说，不管你的汽车使用的是手工的转向盘，还是动力的转向盘，或者是齿轮转向盘，方向盘的工作效果都是一样的：就是把方向盘往左转，车就会左转，不管使用的是什么样子的转向盘。这种统一接口的好处就是一旦我们知道了如何掌控方向盘，我们就可以开什么类型的车。

同样的原理也可以应用在编程上。比如，考虑一个栈，程序可能需要针对三种不同数据类型的栈。一个是针对整型数的，一个是针对浮点型数据的，一个是针对字符的。在这种情况下，栈的实现算法是一致的，只是存储的数据类型是不一样的。在非面向对象的语言中，我们必须创建三个不同的栈的程序，每一个都是用不同的名字。然而，在 C++，通过多态，我们就可以创建一个通用的栈，三种不同的数据类型都可以存储。这样，一旦知道了栈是如何使用的，我们就等于知道了三种不同数据类型的栈的用法。

通常，多态的概念经常被表述如下：一个接口，多种方法。意思是说可以通过对一组相关的活动定义一个通用的接口。多态性通过一个接口来阐明动作的通用性，可以帮助我们减少程序的复杂性。编译器针对应用的情况来选择具体应该是那个动作（方法）。程序员不必手工来做这种选择。我们只需要记住并使用这个通用的接口就可以了。

继承

继承是一个对象可以获取到另外一个对象的属性的过程。继承非常重要，因为它支持了层次化的分类。仔细想想，大多数的知识都是通过层次化的分类来管理的。例如：红色的甜苹果是属于苹果分类的，而苹果分类又是水果分类的一种，水果又是一个更大的类：食物的一种。也就是说，食物有一些特质，比如可以食用，有营养等等，逻辑上来说它的子类水果也是有的。除了这些特质，水果还有特殊的特质，比如多汁，香甜等等，正是这些特质使得水果和其它的食物有所区分。苹果类定义了苹果特有的特质，比如生长在树上，非热带的等等。一个红色香甜的苹果应该继承了它的父类的所有特质，同时也定义了一些使得它和别的苹果区分开来的特质。

如果没有层次化的分类，每一个对象都必须显示地定义自己的所有特性。通过使用继承，一个对象只

需要定义那些使得它在整个类中显得独特的那些特质。它可以从父类那里继承所有通用的特质。因此正是继承机制使得一个对象成为一个通用类的实例成为可能。

专家答疑

问：

文中说到面向对象的编程是一种有效的进行大程序管理的方法。但是面向对象的编程似乎会增加相对小程序的开销。就 C++而言，这点是真是假？

答：

假。理解 C++的关键一点是使用 C++可以编写面向对象的程序，但是并不是说必须写面向对象的程序。这一点是 C++和 Java、C#的一个很重要的不同点。Java 和 C#采用严格的对象模型，所以每个程序，不管其多小，都是面向对象的。C++也提供了该选项。更重要的是，在运行的时候，C++的面向对象特性是透明的，因此，如果存在的话，增加的开销也是很少的。

练习：

1.说出面向对象编程的原则。

2.C++中，封装的最基本的单元是什么？

3.C++中，子程序通常被称作什么？

答案：

1.封装，多态和继承是面向对象编程的原则。

2.C++中，类是封装的基本单元。

3.C++中，子程序通常被成为函数。

必备技能 1.4：第一个简单的程序

现在我们开始编程了。我们通过编译和运行下面的一个简短的 C++程序开始：

```
/*
    这是一个简单的 C++程序.
    文件名称为 Sample.cpp.
*/
#include<iostream>
using namespace std;
//C++程序都是从 main()函数开始的

int main()
{
    cout<<"C++ is power programming.";
    return 0;
}
```

我们将按照下面的三个步骤来进行

1.键入程序

2.编译程序

3.运行程序

在开始之前，让我们先复习两个术语：源代码和目标代码。源代码就是人类可读的程序，它是存储在文本文件中的。目标代码是由编译器生成的程序的可执行形式。

键入程序

本书中出现的程序都可以从 Osborne 的网站上获取到：www.osborne.com。然而，如果你想手工键入这些程序也是可以的。手工键入这些程序通常可以帮助我们记忆这些关键的概念。如果你选择手工键入这些程序，你必须有一个文本编辑器，而不是一个字处理器。字处理器通常存储带有格式的文本信息。这些信息会使得 C++编译器不能正常工作。如果你使用的是 windows 平台，那么你可以使用记事本程序，或者其它你喜欢的程序编辑器。

存储源代码的文件名称从技术上来说是可以任意取的。然而，C++程序文件通常采用.cpp 的扩展名。因此，你可以任意命名 C++的程序文件，但是它的扩展名应该是.cpp。针对这里的第一个例子，源文件取名为 Sample.cpp。本书中的其它程序，你可以自行选择文件名。

编译程序

怎样编译 Sample.cpp 取决于你的编译器和你采用了什么样子的编译选项。更进一步来说，许多编译器，例如你可以自由下载的微软的 Visual C++Express Edition，都提供两种不同的编译方式：命令行编译和集成开发环境。因此，这里不可能给出一个通用的编译 C++程序的操作。你必须查看与你的编译器相关的操作指南。

如果你采用的是 Visual C++，那么最简单的编译和运行本书中的程序的方式就是采用 VC 中提供的命令行。例如,使用 Visual C++来编译 Sampler.cpp，你可以采用这样的命令行：

```
C:\...>cl –GX Sample.cpp
```

其中-GX 是增强编译选项。在使用 Visual C++命令行编译器之前，你必须先执行 visual C++提供的批处理文件 VCVARS32.BAT。Visual Studio 同时也提供了一个很方便使用命令行的方式：可以通过任务栏的开始|程序|Microsoft Visual Studio 菜单中的工具列表中的 Visual Studio Command Prompt 菜单来激活命令行。

C++编译器的输出是一个可以执行的目标代码。在 windows 环境下，可执行文件的名字和源码文件的名字相同，但是扩展名为.exe。因此，Sample.cpp 的可执行文件为 Sample.exe。

运行该程序

C++程序编译好之后就可以运行了。既然 C++编译器的输出为可执行的目标代码，那么运行这个程序，只需要在命令行提示符下键入这个程序的名字即可。例如,运行 Sample.exe 时，采用如下的命令行

```
C:\...>Sample
```

运行时，程序将显示下面的输出：

```
C++ is power programming.
```

如果你使用的是集成开发环境，那么你可以从菜单中选择 Run 来运行这个程序。运行的方式也和编译器相关，这点请参考你使用的编译器的操作指南。针对本书中的程序，通常从命令行来编译和运行会简单

一些。

最后一点：本书中的程序都是基于控制台模式的，而不是基于 windows 模式的。也就是说，这些程序是运行在命令行提示符下的。C++对 windows 编程也是很内行的。实际上，它是 windows 开发时最常用的语言。然而，本书中所有程序都没有使用 windows 的图形化用户界面。这一点的原因也很好理解：windows 编程本质上是很巨大和复杂的，即使是创建一个最小的 windows 框架程序也需要 50 至 70 行代码，而编写能够展示 C++特性的 windows 程序也需要几百行代码。相比之下，基于控制台的程序更短小，通常都是采用基于控制台的程序来学习的。一旦你掌握了 C++，你就能够很轻松地把学习的知识应用到 windows 编程上了。

逐行剖析第一个简单的程序

尽管 Sample.cpp 相当短小，但是它却涵盖了几个 C++中常用的关键特性。让我们近距离地研究一下这个程序的每个部分。程序以

```
/*
    这是一个简单的 C++程序.
    文件名称为 Sample.cpp.
*/
```

开始，这点是很常见的。就像其它大多数编程语言一样，C++允许程序员对程序的源码进行注释。注释的内容会被编译器忽略掉。注释的目的是给所有读源码的人描述或者解释该程序。就本例子来讲，注释解释了该程序的功能。在更复杂的程序中，注释可以被用来解释程序的每个部分是为什么这样做，以及如何做才能以完成预期的功能。换句说，你可以对程序的功能提供一种详细的描述。

在 C++中，有两种注释的方式。刚才看到的叫做多行注释。这种类型的注释以/*开始，以*/结束。任何处于这两个注释符号之间的内容都会被编译器忽略掉。多行注释可以是一行或者多行的。第二种注释会在后续的程序中看到。

接下来的代码是：

```
#include<iostream>
```

C++语言定义了几个头文件，这些头文件中定义了或者必要的或者有用的信息。该程序就需要头文件中的 iostream 来支持 C++的输入/输出系统。该头文件是由编译器提供的。程序中通过#include 来包含头文件。在本书后续章节中，我们会学习到更多的关于头文件和为什么要包含头文件的知识。

接下来的代码是：

```
using namespace std;
```

这行告诉编译器使用 std 命名空间。命名空间是相对较新的增加到 C++里面的特性。本书后续会对命名空间进行详细的讨论，这里给出一个简单的描述。一个命名空间创建了一个声明域，其中可以放置各种程序元素。在一个命名空间中声明的元素和另外一个命名空间中声明的元素是相互分开的。命名空间可以帮助我们组织大型程序。using 语句通知编译器我们要使用 std 这个命名空间。这是声明了整个标准 C++库的命名空间。通过使用 std 这个命名空间，我们访问标准库。（因为命名空间相对比较新，旧的编译器可能不支持。如果你使用的是旧的编译器，请参见附录 B，其中描述了简单的应急措施）

程序的下一行是：

```
//C++程序都是从 main()函数开始的
```

这一行中使用到了 C++中的第二种注释方式：单行注释。单行注释以//开始，在一行的末尾结束。通常情况下，针对较多的，详细的注释使用多行注释，简单的注释一般使用单行注释。当然，这个也完全是

个人编程风格的问题。

接下来的一行代码，正如上面的注释一样，是程序执行的入口：

```
int main()
```

所有的 C++程序都是由一个或者多个函数组成。正如前面解释的那样，一个函数就是一个子程序。每一个 C++函数都必须有个名字，任何 C++程序都必须有一个 main()函数，就像本例子中的一样。main()函数是程序执行的入口，通常也是程序结束的地方。**从技术上来讲，一个 C++程序通过调用 main()函数而开始，大多数情况下，当 main()函数返回的时候，程序也就结束了。**该行中的一对括号表示 main()函数代码的开始。main()函数前面的 int 表明了该函数的返回值为 int 类型。后面会介绍到，C++支持几种内建的数据类型，其中就有 int。它代表的是一个整型数。

接下的一行是：

```
cout<<"C++ is power programming.";
```

这是一条控制台输出语句。它输出：C++ is power programming.到计算机显示器上。输出是通过使用输出运算符<<来完成的。<<运算符把它右侧的表达式输出到左侧的设备上。cout 是一个预先定义好的标识符，它代表了控制台的输出，通常就是指计算机的显示器。因此，这句程序就是把信息输出到计算机的显示器上。请注意，这句程序是以分号结束的。实际上，所有的 C++语句都是以分号结束的。信息“C++ is power programing.”是一个字符串。在 C++中，一个字符串就是由双引号引起来的一个字符的序列。在 C++中，字符串是会经常用到的。

下一行是：

```
Return 0;
```

该行代码终止了 main()函数，并且返回 0 给调用进程（调用进程通常就是操作系统）。对于大多数的操作系统来说，返回值 0 就意味着程序正常终止。其它返回值则代表程序因为产生了错误而终止。return 是 C++的关键字之一，用它来表示函数的返回值。所有的程序在正常终止的情况下（也就是没有出现错误的情况下）都应该返回 0。

程序最后的大括号表示程序到此结束。

处理语法错误

根据以前的编程经验，我们知道，编程时很容易出现键入错误的情况的。庆幸的是，如果我们在键入程序的时候输入错误了，编译器在编译的时候会报告语法错误信息的。大多数的 C++编译器都试图正确理解我们编写的程序。正是由于这个原因，编译器报告的错误并不一定都反映了问题出现的原因。例如，在前面的程序中，如果我们遗忘了 main()后面的大括号，编译器则会报告说 cout 语句有语法错误。当编译器报告语法错误信息的时候，我们就应该检查最新写的那几行代码，看看是否有错误。

专家答疑

问：

除了错误信息以外，编译器还会提供告警信息。告警和错误有什么区别，我应该关注那类信息？

答：

除了报告致命的错误信息以外，大多是的编译器都会上报几种告警信息。错误信息代表的是程序中出现了明确的错误，比如忘记键入分号等。告警信息则是指出现了从技术上讲是正确的，但是值得怀疑的地方。由程序员来决定此处是否正确。

告警可以用来报告诸如使用了已经被废弃的特性或者无效的构成等。通常来讲，我们可以选择那些告警是可见的。本书中的程序是符合标准 C++的，因此只要输入无误，是不会出现令人烦恼的告警信息的。

针对本书中的示例程序，你可能想使用编译器缺省的错误报告。然而，你可以查阅自己编译器的相关文档以明确编译器提供哪些选项。很多编译器都可以帮助我们检查出潜在的错误，不至于引起大麻烦。了解编译器的错误报告机制是很值得的。

练习：

1.C++程序是从哪里开始执行的？

2.cout 是做什么用的？

3.#include<iostream>是用来做什么的？

答案：

1.C++程序从 main 函数开始执行。

2.cout 是预先定义好的标识符，它连接到控制台输出。

3.它用来包含头文件<iostream>，可以支持输入/输出。

必备技能 1.5：第二个简单的程序

变量是程序最基本的构成元素。一个变量就是一个被命名的，可以被赋值的内存位置。进一步来说，变量的取值在程序的执行过程中是可以改变的。也就是说变量的内容是可变的，而不是固定的。

下面的程序创建了一个变量叫做 length，赋值 7，并在显示器上输出信息“The length is 7”。

```
//使用一个变量
#include<iostream>
using namespace std;

int main()
{
    int length; //这行是声明一个变量
    length=7; //给变量赋值为 7
    cout<<"The length is ";
    cout<<length; //这行输出 7，也就是输出变量 length 的值
    return 0;
}
```

正如前文所提到的，C++程序的命名是可以任意的。因此，键入这个程序后，你可以选择一个自己喜欢的文件名字来保存该程序。当然，文件可以取名为 VarDemo.cpp。

这个程序引入了两个概念。第一，语句

```
int length; //这行是声明一个变量
```

声明了一个变量，叫做 length，它的类型为整型。在 C++中，所有的变量都必须在使用之前进行声明。而且，变量所能存储的值的类型也必须指定，这叫做变量的类型。所以上面的代码声明的变量 length 是用来存储整型数的。整型数的取值范围为-32768 到 32767。在 C++中，欲声明一个整型的变量，就在它的名称前加上 int。后面会学习到 C++支持更多的内置的变量类型。（还可以自己创建自己的数据类型）

接下来的一行中用到了第二个新的特性：

```
length=7; //给变量赋值为 7
```

正如注释所解释的那样，这句代码给变量 length 赋值为 7。在 C++中，赋值运算符就是一个单等号。它把右侧的取值拷贝到左侧的变量中。在该赋值语句之后，变量 length 的取值将为 7。

接下来的语句输出了变量 length 的值：

```
cout<<length; //这行输出 7，也就是输出变量 length 的值
```

通常来讲，如果我们想要输出一个变量的取值，只要把它放置在 cout 语句中<<的右侧即可。针对例子中的情况，因为 length 的取值为 7，因此输出的值就会是 7。在继续学习下面的内容之前，我们可以尝试给 length 赋于其它的值，并观察输出的结果。

必备技能 1.6：使用运算符

和大多数其它的语言一样，C++支持全部的算术运算符，以便能够处理程序中的数字。这其中就包括：

```
+加法
-加法
*乘法
/除法
```

在 C++中，这些运算符的用法和代数中的用法是一样的。

下面的程序使用*运算符来根据长度和宽度计算矩形的面积。

```
//使用一个运算符
#include <iostream>
using namespace std;

int main()
{
    int length ; // 这行定义了一个变量
    int width ; // 这行定义了另外的一个变量
    int area; // 这行也是定义变量
    length = 7; // 给变量 length 赋值为 7
    width = 5; // 给变量 width 赋值为 5
    area = length*width; //计算面积，把 length 和 width 的乘积赋值给 area
    cout<<"The area is ";
    cout<<area; //输出 35
    return 0;
}
```

这个程序声明了三个变量：length，width 和 area。给变量 length 赋值为 7，width 赋值为 5。然后计算他们的乘积，并把结果赋值给变量 area。这个程序的输出如下：

```
The area is 35
```

在这个程序中实际没有必要声明变量 area。可以按照下面的示范重写一下刚才的程序：

```
// 简化版本的面积计算程序
#include <iostream>
using namespace std;
int main ()
{
```

```
    int length; //声明一个变量
    int width; // 声明另外的一个变量
    length =7; //给 length 赋值为 7
    width=5; //给 width 赋值为 5a
    cout << "The area is ";
    cout << length * width; //输出 35
    return 0;
}
```

在这段代码中，面积是在 cout 语句中通过把 length 和 width 做乘法运算而得到的，然后把结果输出到显示屏幕上。

在继续学习之前，我们还应该指出，在同一个声明变量的语句中，我们是可以声明两个或者多个变量的。只需要用逗号把它们分开即可。例如，可以通过下面的方式声明三个变量 length,width,area：

```
int length,width,area; //使用一条语句声明全部的变量
```

在专业的代码中，通过一条语句声明两个或者多个变量是非常普遍的。

练习：

1.变量在使用前是否必须先声明？

2.如何给变量 min 赋值 0？

3.在一条声明语句中是否可以声明多个变量？

答案：

1.是的，C++中的变量在使用前都必须声明。

2.min=0;

3.是的。在一条声明语句中可以声明多个变量。

必备技能 1.7：从键盘读取输入

前面的程序中使用的都是显示指定的数据。例如，计算面积的程序计算的是长为 7，宽为 5 的矩形的面积，矩形的尺寸本身就是程序的一部分。然而，不管矩形的尺寸是多少，其面积的计算方法都是一样的。因此，如果能够提示用户从键盘输入矩形的尺寸，然后计算矩形的面积的话，那么这个程序将够有用一些。

我们可以使用>>运算符来使用户从键盘输入数据到程序中。这就是 C++中的输入运算符。通常使用下面的形式从键盘获取数据

```
cin>>var;
```

其中，cin 是另外一个预先定义好的标识符。它代表控制台输入，这是 C++自动支持的。缺省情况下，cin 是和键盘绑定的，它也可以被重定向到其它的设备上。var 代表接收输入的变量。

下面重写计算面积的程序，允许用户输入矩形的尺寸：

```
/*
    用来计算矩形面积的交互式程序
*/
#include <iostream>
using namespace std;

int main()
```

```
{
    int length; // 声明一个变量
    int width;  // 声明另外一个变量

    cout << "Enter the length:";
    cin >> length; //从键盘输入长度

    cout << "Enter the width:";
    cin >> width; //从键盘输入宽度

    cout << "The area is ";
    cout << length * width; //输出面积

    return 0;
}
```

运行的结果可能如下：

```
Enter the length: 8
Enter the width: 3
The area is 24
```

请注意下面的几行：

```
cout<<"Enter the length:"
cin>>length; //从键盘输入长度
```

cout 语句提示用户输入数据。cin 语句读取用户输入的数据，并把值存储在变量 length 中。于是，用户键入的数值（就本例中的程序，用户必须输入一个整型数）就被存放在了>>右侧的变量中（本例中就是 length）。在执行完毕 cin 语句后，变量 length 存放的就是矩形的长度（如果用户键入的是非数字，变量 length 的值将会是 0）。提示用户输入宽度和从键盘读取矩形长度的语句工作原理是一样的。

一些输出选项

到目前为止，我们一直使用的都是 cout 的最简单的形式。然而，cout 可用来输出更复杂的语句。下面是两个有用的技巧。首先，我们可以使用一个 cout 语句输出多条信息。比如，在计算面积的程序中，我们使用下面的两行来输出面积：

```
cout<<"The area is ";
cout<<length*width;
```

这两行代码可以使用下面更方便的语句来表达：

```
cout<<"The area is "<<length*width;
```

这种方式在同一个 cout 语句中使用了两个输出运算符，这将在输出字符串"The area is"后接着输出面积。通常情况下，我们可以在一条输出语句中连接多个输出运算符，每个输出项单独使用一个运算符即可。

第二个技巧，到目前为止我们还没有换行输出，也就是回车。但是不久我们就需要这样输出了。在字符串中我们使用"\n"表示换行，试试下面的程序就可以看到"\n"的效果了。

```
/*
    这个程序演示了\n 的用法，可以输出换行
*/
#include <iostream>
using namespace std;
int main()
{
    cout << "one \n";
    cout << "two \n";
    cout << "three ";
    cout << "four ";
    return 0;
}
```

这段程序的输出如下：

```
one
two
three four
```

换行字符可以被放置在字符串中的任意位置，并不一定是放置在最后。在理解了换行字符的作用后，就可以自己写程序看看结果了。

练习：

1.C++中的输入运算符是哪个？

2.默认情况下，cin 是和那个设备进行绑定的？

3.\n 代表什么？

答案：

1.输入运算符是>>

2.cin 默认是和键盘绑定的。

3.\n 代表换行字符。

其它的数据类型

在前面的程序中，我们使用的都是 int 类型的变量。int 类型的变量只能用来存储整型数。当需要存储小数的时候，int 类型的变量就不能使用了。比如，一个 int 类型的变量可以用来存储值 18，但是不能用来存储值 18.3。幸运的是，int 类型只是 C++中定义的几种数据类型之一。C++定义了两种浮点类型来表示小数：float 和 double 类型，它们分别代表单精度和双精度的小数。其中，double 可能是最常用的了。

可以采用类似下面的方式来声明一个 double 类型的变量：

```
double result;
```

这里，result 是变量的名称，它的类型是 double 类型的。因为它的类型是浮点类型，因此可以被用来存放诸如 88.56 或者-107.03 之类的数据。

为了更好地理理解 int 和 double 类型的区别，可以试一试下面的程序：

```
/*
```

```
    这个程序展示了 int 类型和 double 类型的区别
*/
#include <iostream>
using namespace std;
int main()
{
    int ivar; //声明一个整形的变量
    double dvar; //声明一个浮点型的变量

    ivar = 100; //给 ivar 赋值 100
    dvar =100.0; //给 dvar 赋值 100.0

    cout<< "Original value of ivar: " << ivar << "\n";
    cout<< "Original value of dvar: "<< dvar << "\n";
    cout<< "\n"; //打印一个空行

    //各自都除以 3
    ivar=ivar/3;
    dvar=dvar/3.0
    cout << "ivar after division: " << ivar << "\n";
    cout << "dvar after division: " <<dvar <<"\n"

    return 0;
}
```

程序的输出如下：

```
Original value of ivar:100
Original value of dvar:100
ivar after division:33
dvar after division:33.3333
```

从上面的例子可以看出，当 ivar 除以 3 的时候，得到的结果是 33，小数部分丢失了。而 dvar 除以 3 后，小数部分是被保留的。

该程序中还有一个新的需要注意的地方：

```
cout<<"\n"//输出一个空行
```

这句将输出一个空行。可以在需要任何需要输出空行的地方使用该语句。

专家答疑

问：

为什么 C++中用两种不同的数据类型来分别表示整型数和浮点数了？也就是说，为什么不是所有的数据都仅仅使用一个类型了？

答：

C++提供了不同的类型是为了程序的效率更高。比如，整形数的运算要比浮点数的运算快。因此，如果不需要小数，就没有必要引入 float 或者 double 类型带来的开销。还有就是，各种类型的数据在存储的时候所需的内存的大小也是不一样的。通过支持不同的数据类型，C++使得我们可以最好地利用系统的资源。最后，一些算法是需要一些特定类型的数据的。C++提供了大量的内置类型以提供最好的灵活性。

项目 1-1 英尺到米的转换

虽然前面的几个程序展示了 C++语言的几个重要特性，但是它们并不实用。尽管到目前为止我们对 C++的学习并不多，但是我们仍然可以利用所学习的知识来创建一个实用的程序。在该项目中，我们创建一个程序来把英尺转换成米。该程序提示用户输入英尺数据，然后显示出转换后的米的数据。

一米大约等于 3.28 英尺，因此我们需要使用浮点数。为了完成转换，程序中声明了两个浮点类型的变量，一个用来存储英尺的数据，一个用来存储米数据。

步骤：

1.创建一个 C++文件，命名为 FtoM.cpp。（注意：在 C++中文件的名字是可以任意的，你可以取任何自己喜欢的名字。）

2.文件以下面的内容开始。这些内容解释了该程序的功能，包含了 iostream 头文件，指明了 std 命名空间。

```
/*
    Project 1-1   该程序实现从英尺到米的转换
    程序文件命名为 FtoM.cpp
*/
#include <iostream>
using namespace std;
```

3.main()函数中声明变量 f 和 m：

```
int main()
{
    double f; // 用来存储英尺数的变量
    double m; // 用来存储米数的变量
```

4.增加输入英尺数据的代码：

```
    cout << "Enter the length in feet:";
    cin>>f; // 读取英尺数据
```

5.增加进行转换和输出的代码：

```
     m= f / 3.28; //转换成米
     cout << f << " feet is " << m << " meters.";
```

6.以下面的代码结束程序：

```
      return 0;
}
```

7. 程序完成后应该是下面的这个样子：

```
/*
    Project 1-1   该程序实现从英尺到米的转换
    程序文件命名为 FtoM.cpp
```

```
*/
#include <iostream>
using namespace std;
int main()
{
    double f; // 用来存储英尺数的变量
    double m; // 用来存储米数的变量

    cout << "Enter the length in feet: ";
    cin  >> f; // 读取英尺数据

    m = f / 3.28; // 转换成米
    cout << f << " feet is " << m << " meters.";

    return 0;
}
```

8. 编译并运行该程序，举例的输出结果如下：

```
Enter the length in feet: 5
5 feet is 1.52439 meters.
```

9. 可以尝试着输入其它的数据，也可以尝试修改成把米转换成英尺。

练习：

1.在 C++中整型类型的关键字是什么？

2.double 是什么意思？

3.怎样才能输出新的一行？

答案：

1.整形数据类型为 int

2.double 是用来表示双精度的浮点数的。

3.输出新的一行，使用\n.

必备技能 1.8：两条控制语句

在一个函数内部，语句是按照从上到下的顺序执行。然而，我们可以通过使用 C++支持的各种程序控制语句来改变程序的顺序执行方式。尽管我们会在后文中仔细研究控制语句，在这里我们还是要简单介绍两种控制语句，我们可以使用它们来写几个简单的示例程序。

if 语句

我们可以使用 C++的条件语句选择性地执行程序的部分代码：if 语句。C++中的 if 语句和其它程序语言中的 if 语句是很类似的。例如，在语法上 C++中的 if 语句和 C，Java，C#中的都是一样的。它的最简单的形式如下：

```
if ( condition ) statement;
```

这里的 condition 是一个待评估的条件表达式，它的值或者是 true（真）或者是 false（假）。在 C++中，true 就是非零的数值，而 false 就是 0。如果 condition 为真，则 statement 部分将会被执行。如果 condition 为假，statement 部分将不会被执行。例如，下面的代码片段将在显示器上显示出片语：10 is less than 11，因为 10 是小于 11 的。

```
if( 10 < 11 ) cout << " 10 is less than 11";
```

再考虑下面的代码片段：

```
if ( 10 > 11 ) cout << " this dose not display";
```

在上面的片段中，10 并不是大于 11 的，所以 cout 语句并不会被执行。当然，if 语句中的条件表达式中不一定都必须使用常量。它们也可以是变量。

C++中定义了全部的关系运算符，它们可以被用在条件表达式中。如下：

运算符	含义
<	小于
<=	小于或者等于
>	大于
>=	大于或者等于
==	等于
!=	不等于

注意，等于是两个等号。

下面是一个示例程序，它展示了 if 语句的用法：

```
// if 的用法示例
#include <iostream>
using namespace std ;
int main()
{
    int a,b,c;
    a = 2;
    b = 3;

    if ( a < b ) cout << " a is less than b \n";   //这里是一条 if 语句
    //下面的语句将不能显示任何东西
    if ( a == b ) cout << " you won't see this\n"
    cout << \n;

    c = a - b; //c 的值为-1
    cout << " c contains -1\n";
    if ( c >= 0 ) cout << "c is no-negative\n";
    if ( c < 0 ) cout << " c is negative\n";
    cout <<"\n";

    c = b- a; // c 的值现在为 1
```

```
        cout << " c contains 1\n";
        if ( c>= 0 ) cout << "c is non-negativa\n";
        if ( c < 0 ) cout << "c is negative\n";

        return 0;
    }
```

这个程序的输出如下：

```
a is less than b

c contains -1
c is negative

c contains 1
c is non-negative
```

for 循环

通过循环，我们可以反复执行一系列的语句。C++提供了多种不同的方式来支持循环。这里我们将要看到的是 for 循环。如果你很熟悉 Java 或者 C#，你会很高兴地看到，C++中的 for 循环和这两个语言中的 for 循环是一样的。for 循环最简单的形式如下：

```
for(initialization ; condition ; increment )
```

这里 initialization 用来设置循环控制变量的初始值。condition 是每次重复执行的时候需要进行的检查条件，只要条件为真，循环就一直继续。increment 是一个表达式，它表示每次循环的时候，循环的控制变量时如何递增的。

下面的程序演示了 for 循环的使用方法。它在屏幕上输出数字 1 到 100。

```
// A program that illustrates the for loop.
#include <iostream>
using namespace std;
int main()
{
    int count;

    for ( count = 1; count<=100;count=count+1)
        cout << count <<" ";

    return 0;
}
```

在上面的循环中，count 初始化为 1。循环每执行一次，条件 count<=100 就会被检查一次。如果条件为真，则输出 count 的取值，并且 count 增加 1。当 count 的值大于 100 的时候，检测的条件就为假了，循环终止了。在专业的代码中，是永远不会看到类似下面的语句的：

```
count = count + 1;
```

因为 C++中包含了一个特殊的自增运算符来更有效的完成上面的功能。这个自增运算符就是++，也就是两个连续的加号。自增运算符使得运算数的值增加 1。前面的 for 语句通常会被写成下面的形式：

```
for(count=1; count <= 100; count++) cout << count << " ";
```

本书的后续章节将都会采用这种书写的形式。

C++中同时提供了自减的运算符：--，它使得运算数的值减 1。

练习：

1.if 语句是用来做什么的？

2.for 语句是用来做什么的？

3.C++中有哪些关系运算符？

答案：

1.if 语句是 C++中的条件语句。

2.for 语句是用来做循环用的。

3.C++中的关系运算符有：==，!=,<，>，<=,>=

必备技能 1.9：使用代码块

C++中一个基本的元素就是代码块。一个代码块由两个或者多个语句组成，由一对花括号括起来的。代码块可以作为一个逻辑单元出现在任何单条语句可以出现的地方。例如，代码块可以被使用在 if 语句或者 for 循环中：

```
if ( w < h )
{
    v = w * h;
    w = 0;
}
```

这里，如果 w 小于 h，则代码块中的两条语句都会被执行。因此，代码块中的两条语句形成了一个逻辑单元，要么两条语句都会被执行，要么两条语句都不会被执行。凡是需要把两条或者多条语句在逻辑上进行关联的地方，我们都可以使用代码块。代码块使得许多算法实现起来更加清晰和有效。

下面的程序中使用了代码块来防止除数为 0。

```
//Demonstrate a block of code
#include <iostream>
using namespace std;
int main()
{
    double result,n,d;

    cout << "Enter value: ";
    cin >> n;

    cout << "Enter divisor: "
    cin >> d;
    //the target of this if is a block
```

```
    if ( d!= 0 )
    {
        cout << "d does not queal zero so division is OK" << "\n";
        result = n / d;
        cout << n << " / " << d << " is " << result;
    }

    return 0;
}
```

一个示例的输出结果如下：

```
Enter value: 10
Enter divisor: 2
d does not queal zero so division is OK
10 / 2 is 5
```

在上面的这个例子中，if 语句的目标不是一条语句，而是一个代码块。如果 if 语句的控制条件为真，代码块中的三条语句都会被执行。可以尝试输入除数为 0，并观察程序的输出结果。此时，代码块中的三条语句都不会被执行。

在本书的后面的的章节中，我们会看到代码块还有别的属性和用法。然而，代码块存在的主要原因是它能够创建在逻辑上不可分割的独立单元。

专家答疑

问：

使用代码块是否会降低运行时的效率？也就是说代码块中的大括号是否会在代码执行的过程中消耗运行时间？

答：

不会的。代码块在运行的时候并不会增加任何的开销。实际上，由于它能简化一些特性算法的编码，使用代码块通常会增加运行的速度和效率。

分号和位置

在 C++中，分号代表着语句的结束。也即是说，每一个单独的语句都必须以分号结束。通过前面的介绍，我们知道，代码块是在逻辑上关联的一组语句的集合，它由一对花括号括起来。代码块不是以分号作为结束的。既然代码块是一组语句的集合，其中每条语句都是以分号结束，所以代码块不是以分号结束是有意义的。代码块的结束是通过}来标记的。

C++中，并不是一行的结束就代表着一条语句的结束，只有分号才能代表语句的结束。所以，我们把语句写在了哪一行并不是很重要。例如，在 C++中：

```
x = y;
y = y + 1;
cout << x << " " << y;
```

和下面的代码是一样的：

```
x = y; y = y + 1; cout << x << " " << y;
```

更进一步，语句的单独元素也可以被放置在不同的行中。例如，下面的代码也是可以接受的：

```
cout << "This is a long line. The sum is：" << a + b + c +
                d + e + f;
```

采用上面的方式避免一行中写过多的代码可以增加代码的可阅读性。

缩进

在前面的例子中，可以看到一些语句采用了缩进的格式。C++是一个很自由的语言，也就是说，在一行中，把相关的代码放置在了行的什么地方并不要紧。然而，人们已经形成了常用的和可以接受的缩进风格，方便阅读程序。本书将遵循这样的缩进风格。并建议读者也这样做。在这种风格中，针对每个{后面的代码都需要缩进一个级别，在}之后代码的缩进格式需要向前提升一个级别。还有一些语句将采用另外的缩进方式，本书后续会进行描述。

练习：

1.如何创建代码块？它是用来做什么用的？

2.在 C++中语句块是以_____结束的。

3.所有的 C++语句都必须在同一行开始和结束，对吗？

答案：

1.代码块是以{开始，以}结束的。它是用来创建逻辑单元的。

2.分号。

3.错误。

项目 1-2 生成一张从英尺到米的转换表

这个项目将显示一张从英尺到米的转换表，其中用到了 for 循环，if 语句和代码块。表格的起始为一英尺，终止为 100 英尺。每输出 10 英尺的转换表后，输出一个空行，这个是通过使用变量 counter 来实现的。变量 counter 是用来表示行数的，请注意它的用法。

步骤：

1.创建一个新的文件，命名为 FtoMTable.cpp。

2.文件中键入下面的程序：

```
#include <iostream>
    using namespace std;
    int main()
    {
        double f; //  用来存储英尺长度
        double m; //用来存储转换后的米的值
        int counter;

        counter = 0; //  用于统计行数，变量初始化为 0
```

```
    for ( f =1.0; f <= 100; f++)
    {
        m = f /3.28; //英尺转换为米
        cout << f << " feet is " << m << " meters.\n";

        //循环每次都要把行数增加 1
        counter++;

        //每 10 行输出一个空行
        //如果行数为 10 了，则需要输出一个空行
        if ( counter == 10 )
        {
            cout << "\n"; //输出空行
            counter = 0;  //重置行数计数器
        }
    }

    return 0;

}
```

3.注意我们是如何使用 counter 变量来实现每 10 行输出一个空行的。在 for 循环外该变量被初始化为 0。在 for 循环体内部，每做一次英尺到米的转换，counter 就增加 1。当 counter 增加到 10 的时候，就输出一个空行，然后重置 counter 的值为 0，重新开始上面的过程。

4.编译并运行上面的程序。下面是输出的一部分。注意程序输出的小数部分并不是很整齐。

```
1 feet is 0.304878 meters.
2 feet is 0.609756 meters.
3 feet is 0.914634 meters.
4 feet is 1.21951 meters.
5 feet is 1.52439 meters.
6 feet is 1.82927 meters.
7 feet is 2.13415 meters.
8 feet is 2.43902 meters.
9 feet is 2.7439 meters.
10 feet is 3.04878 meters.

11 feet is 3.35366 meters.
12 feet is 3.65854 meters.
13 feet is 3.96341 meters.
14 feet is 4.26829 meters.
15 feet is 4.57317 meters.
16 feet is 4.87805 meters.
```

```
17 feet is 5.18293 meters.
18 feet is 5.4878 meters.
19 feet is 5.79268 meters.
20 feet is 6.09756 meters.

21 feet is 6.40244 meters.
22 feet is 6.70732 meters.
23 feet is 7.0122 meters.
24 feet is 7.31707 meters.
25 feet is 7.62195 meters.
26 feet is 7.92683 meters.
27 feet is 8.23171 meters.
28 feet is 8.53659 meters.
29 feet is 8.84146 meters.
30 feet is 9.14634 meters.

31 feet is 9.45122 meters.
32 feet is 9.7561 meters.
33 feet is 10.061 meters.
34 feet is 10.3659 meters.
35 feet is 10.6707 meters.
36 feet is 10.9756 meters.
37 feet is 11.2805 meters.
38 feet is 11.5854 meters.
39 feet is 11.8902 meters.
40 feet is 12.1951 meters.
```

5.可以自己尝试修改上面的程序为每 25 行输出一个空行。

必备技能 1.10：引入函数

C++程序通常是由多个叫做函数的代码块构成的。尽管我们会在第 5 章对函数进行详细的介绍，这里也有必要进行一个简单的介绍。函数的定义如下：一个包含了一条或者多条 C++语句的子程序。

每个函数都有一个名字，通过名字来调用该函数。调用函数的时候，需要在自己的程序的源码中指定函数的名字，后面紧跟着函数的参数。例如，有个函数名字为 MyFunc。如下代码展示了如何调用该函数：

```
MyFunc();
```

当调用一个函数的时候，程序的控制就转移到了被调用函数中，函数中的代码将被执行。当函数中的代码执行完毕后，程序的控制又转回给函数的调用者。因此，函数是为一个程序的其它部分完成相应功能的。

有些函数需要一个或者更多的参数，我们在调用的时候需要传入这些参数给函数。因此，一个参数就是我们传入到函数中的值。在调用函数的时候，参数是放置在括号中的。例如，如果函数 MyFunc()需要一个整型数作为参数，那么下面的代码就是在调用函数的时候传入了参数 2：

```
MyFunc(2);
```

当函数需要两个或者更多参数的时候，这些参数之间用逗号来分隔。在本书中，术语参数列表指的就是用逗号分隔的参数。注意，并不是所有的函数都需要参数。如果函数不需要参数，则括号中为空。

函数可以为调用者返回一个值。并不是所有的函数都需要返回值的，但大多数都是需要返回值的。函数的返回值可以被赋值给调用者中的一个变量，这是通过把对函数的调用放置在赋值语句的右边来实现的。例如，如果函数 MyFunc()返回一个值，那么可以通过下面的方式来调用它：

```
x=MyFunc(2);
```

首先，函数 MyFunc()被调用。当它返回的时候，它的返回值被赋值给 x。还可以在表到式中调用函数。例如，

```
x=MyFunc(2)+10;
```

在这中情况下，函数 MyFunc()的返回值加上 10 后被赋值给 x。通常情况下，当在语句中遇到函数的名字，该函数就会自动被调用，以便获取它的返回值。

复习一下，一个参数就是调用函数的时候传入的值。返回值是函数返回给调用者的值。下面是一个简短的小程序，它展示了如何调用函数。其中使用了 C++中内置的函数，叫做 abs()来计算一个数的绝对值。abs()函数需要一个参数，把这个参数转换成它的绝对值，并返回结果。

```
// use the abs() function
#include <iostream>
#include <cstdlib>
using namespace std;

int main()
{
    int result;
    result = abs(-10); //调用函数 abs(),将其返回值赋值给变量 result
    cout << result;
    return 0;
}
```

这里，传入到 abs()函数中的值为-10。abs()函数在调用的时候接收传入的参数-10，并返回-10 的绝对值，也就是 10 给调用者。这个指被赋值给变量 result。因此，该程序在屏幕上显示 10。

关于上面这个程序需要注意的另外一个地方：它包含了头文件 cstdlib。这是 abs()函数需要的头文件。一旦我们使用了内置的函数，我们必须包含它的头文件。

通常情况下，我们的程序需要两种类型的函数。第一种就是我们自己写的函数，main()就是其中之一。后面，我们会学习到如何自己编写其它的函数。我们会看到，一个真实的 C++程序实际上包含了许多自己编写的函数。

第二种就是编译器提供的函数。前面例子中的 abs()函数就是一个这样的函数。我们自己写的程序通常是即包含自己编写的函数也包含编译器提供的函数。

当在本文中提及函数的时候，本书已经也即将继续使用 C++程序中的传统表达函数的方式：就是函数名字后面跟一对括号。例如有个函数的名字为 getval,在书写的时候会书写成 getval()。本书中的这种写法是为了把变量的名字和函数的名字区分开来。

C++库

正如前面解释的那样，abs()函数是 C++编译器提供的。这个函数是在 C++的标准库中提供的。C++标准库还提供了许多其它的函数。本书中的程序都会用到库函数。

C++的标准函数库中定义了大量的函数。这些函数完成了很多必要的功能，包括输入输出操作，数学运算和字符串处理。当使用库函数的时候，C++编译器自动把这些函数的目标代码链接到我们自己程序的目标代码中去。

因为 C++标准库很庞大，它已经包含了许多我们在自己程序中需要的函数。这些函数就像盖房子时候的砖块，我们仅仅需要把它们装配起来就可以了。我们可以仔细阅读编译器的函数库的文档，看看编译器都提供了那些有用的函数。我们会惊奇地发现这些函数是如此的多样。如果我们自己编写了一个自己反复使用的函数，这个函数也可以存储在库中。

除了提供库函数外，每个 C++编译器还提供了类库，这个就是面向对象的库。然而，在我们学习了类和对象以后，我们才能使用类库。

练习：

1.什么是函数？

2.函数是通过使用它的名字来调用的，对吗？

3.什么是 C++标准函数库？

答案：

1.一个函数就是包含了一条或者多条语句的子程序。

2.正确

3.C++标准库就是所有 C++编译器提供的函数的集合。

必备技能 1.11：C++中的关键字

标准 C++目前定义了 63 个关键字，如表格 1-1 所示。它们和正式的语法一起形成了 C++语言。早期的 C++版本定义了 overload 关键字，但是现在他已经被废弃了。请记住 C++是一个大小写敏感的语言，并且要求所有的关键字都必须是小写的。

asm	auto	bool	break
case	catch	char	class
const	const_cast	continue	default
delete	do	double	dynamic_cast
else	enum	explicit	export
extern	false	float	for
friend	goto	if	inline
int	long	mutable	namespac
new	operator	private	protected
public	register	reinterpret_cast	return
short	signed	sizeof	static

static_cast	struct	switch	template
this	throw	true	try
typedef	typeid	typename	union
unsigned	using	virtual	void
volatile	wchar_t	while	

表格 1-1 C++中的关键字

必备技能 1.12：标识符

在 C++中，一个标识符可以是一个函数、变量或者任何其它用户自定义项目的名字。标识符可以是一个到几个字符的长度。变量的名字可以是以字母表中任意字符开始或者以下划线开始。后面接着字母，数字或者一个下划线。使用下划线可以增强变量名字的可读性，诸如变量 line_count。大写字母和小写字母是有区分的；也就是说，在 C++中，myvar 和 MyVar 是两个不同的名字。还有一条重要的标识符的规则：任何关键字都不能作为标识符的名字。另外预先定义好的标识符，例如 cout 是不受这个限制的。下面是一些合法的标识符实例

Test	x	y2	MaxIncr
up	_top	my_var	simpleInterest23

记住，标识符不能以数字开头。因此 98OK 是无效的标识符。好的编程实践告诉我们，标识符的名字应该能够反映该标识符的含义或者用法。

练习：

1.哪个是 C++中的关键字，for，For 还是 FOR？

2.一个 C++中的标识符可以含有那种类型的字符？

3.标识符 index21 和 Index21 是否是同一个标识符？

答案：

1.for 是 C++中的关键字。在 C++中所有的关键字都是小写的。

2.一个 C++中的标识符可以含有字母，数字和下划线。

3.不是。C++是大小写敏感的语言。

复习题

1.有人说 C++是现代编程的核心。请解释这句话。

2.C++编译器编译出计算机可以直接执行的代码，对吗？

3.面向对象编程的三个主要原则是什么？

4.C++程序从哪里开始执行？

5.什么是头文件？

6.<iostream>是什么？下面的代码是用来做什么的？

```
#include <iostream>
```

7.什么是命名空间？

8.什么是变量？

9.下面哪个/些变量的命名是不合法的？

a.count
b._count
c.count27
d.67count
e.if

10.如何创建单行的注释？如何创建多行的注释？

11.if 语句的常用形式是怎样的？for 循环的常用形式是怎样的？

12.如何创建代码块？

13.在月球上，重力大约为地球的 17%。编写一个程序，显示一张地球的磅重量对应的在月球上的重量。表格从 1 开始到 100 结束，每 25 磅输出一个空行。

14.木星上的一年（就是木星围绕太阳旋转一周需要的时间）大约为地球上的 12 年。编写一个程序把木星年转换成地球年，由用户指定木星年，允许出现小数的年。

15.当调用函数的时候，程序控制会发生什么变化？

16.编写一个程序，用来计算用户输入的 5 个数据的绝对值的平均值，显示其结果。