第 1 周课程简介

开始学习如何使用 C++进行编程之前，读者需要有几样东西：编译器、编辑器以及本书。即使没有 C++编译器和编辑器，读者仍可通过本书学习 C++，但如果能做些练习，将学到更多的知识。

学习编程的最好方法就是编写程序！在每章最后都有作业，其中包括测验和一些练习。请务必花时间回答所有的问题，并尽可能客观地为自己打分。后面的章节都是以前面的内容为基础的，因此继续阅读后面的内容之前，一定要完全理解当前的知识。

C 程序员的注意事项

C 程序员一定很熟悉本书前 5 章的内容。然而，如果要遵循 C++标准，将存在一些细微的差别，请务必浏览这些内容并完成练习，以确保能够快速地进入第 6 章的学习。

本周内容简介

本周简要地介绍编程（尤其是 C++编程）所需的一些知识。第 1 章和第 2 章介绍有关编程和程序流程的基本概念；第 3 章介绍有关变量和常量以及如何在程序中使用数据的知识；第 4 章讨论程序如何根据运行时提供的数据和遇到的条件执行不同的分支；第 5 章介绍函数是什么以及如何使用它们；第 6 章介绍类和对象；第 7 章介绍更深入地探讨有关程序流程的知识。经过本周的学习后，读者将能够编写真正的面向对象的程序。

第 1 章　绪　　论

欢迎使用本书！今天你将迈出成为高级 C++程序员的第一步。

本章将学习：

- 为何 C++是软件开发的标准？
- 开发 C++程序的步骤。
- 如何输入、编译和链接你的第一个 C++程序。

1.1　C++简史

第一代电子计算机诞生于二战期间，用来帮助计算武器弹道数据。从此以后，计算机语言经历了翻天覆地的变化。起初，程序员们使用最原始的计算机指令即机器语言来编程。这些指令是由 0 和 1 组成的很长的字符串。很快，人们就发明了汇编语言，将机器指令映射为人们可以阅读和易于处理的助记符，如 ADD 和 MOV。

随后出现了高级语言，如 BASIC 和 COBOL。这些语言让人们能够用一些类似于单词或句子的源代码来编程，如 Let I=100。这些指令再通过解释器和编译器转换为机器语言。

解释器翻译并执行程序，它读取程序指令（源代码），直接将其转换为操作。

编译器先将源代码转换为中间格式；这通常被称为编译，它生成目标文件。然后，编译器调用链接器，将目标文件组合成为可执行程序。

由于解释器按原样读取代码并当场执行代码，因此程序员使用起来比较方便。当前，大部分解释型程序被称为脚本，解释器被称为脚本引擎。

有些语言（如 Visual Basic 6）将解释器称为运行库；其他语言（如 Visual Basic .NET 和 Java）有另一个组件——虚拟机（Virtual Machine，VM）或运行库。它也是解释器，然而它不是源代码解释器——将人类可读的语言转换为依赖于计算机的机器码；而是对一种编译后的独立于计算机的语言（中间语言）进行解释和执行。

编译器增加了一个步骤：将人类能够理解的源代码编译成机器能够理解的目标代码。这个步骤看似不太方便，但由于将源代码转换为机器语言这样一个耗时的任务已经在编译阶段完成，因此编译型程序的运行速度非常快。由于转换工作已经完成，因此执行程序时无需做这样的工作。

诸如 C++等编译型语言的另一个优点是，可以向没有编译器的用户提供可执行程序。使用解释型语言时，必须有解释器才能运行程序。

C++通常是一种编译型语言，虽然存在一些 C++解释器。和众多其他编译型语言一样，C++以能够生成快速而功能强大的程序著称。

事实上，很多年来，计算机程序员的一个主要目标是编写能够快速执行的简短代码。程序必须短小，因为内存昂贵；同时必须快速运行，因为处理能力也很昂贵。随着计算机的体积越来越小、价格越来越便宜、速度越来越快，内存的价格不断下降，这些优先考虑的因素发生了变化。当前，程序员的时间费用已远远超过了大多数商用计算机的费用。精心编写和易于维护的代码成了编程的首要目标。易于维护是指当程序需求

发生变化时，无需花太大的代价就可以对程序进行扩展和改进。

注意：程序一词通常有两种含义：一种是指程序员编写的指令（源代码），另一种是指可执行软件。这种区分可能会引起巨大的混乱，因此本书尽可能将源代码和可执行文件区分开来。

1.1.1 解决问题

当前，程序员们面临的问题与 20 年前完全不同。在 20 世纪 80 年代，人们编写程序是为了管理和处理大量的原始数据。那时，编写代码的人和使用程序的人都是计算机专业人员。现在，越来越多的人开始使用计算机，其中大部分人对计算机和程序的工作原理知之甚少。人们通常只愿意将计算机作为一种工具来解决商务问题，而无心去钻研它。

具有讽刺意味的是，为了满足这些新用户易于使用的要求，程序变得越来越复杂。那种需要输入神秘的命令，结果却只看到一大串枯燥的数据的时代已一去不复返。当前的程序大都采用“用户友好界面”，其中包括多个窗口、菜单、对话框以及非常形象的标识，对此我们已经非常熟悉了。

随着万维网的发展，计算机跨入了一个新的市场渗透时代：使用计算机的人比任何时候都多，他们对计算机的期望也非常高。万维网的易用性也提高了人们的期望，他们期望程序能够充分利用万维网提供的信息。

在过去的几年中，应用程序的运行平台已扩展到其他设备，不再限于桌面 PC，而被用于手机、个人数字助理（PDA）、平板 PC 和其他设备。

在本书第一版面世后的几年中，为满足用户的需求，程序员编写的程序变得越来越庞大，越来越复杂。因此，对有助于应对这种复杂性的编程技术的需求显得越来越迫切。

随着编程要求的改变，用于编写程序的语言和技术都得到了发展，以帮助程序员应对这种复杂性。虽然完整的发展历史令人神往，但本书只介绍其中的重要组成部分：从过程化编程到面向对象编程的转变。

1.1.2 过程化编程、结构化编程和面向对象编程

直到不久前，计算机程序还被看作是一系列处理数据的过程。过程（函数或方法）是指一组依次执行的指令。数据与过程是分离的，编程技巧是跟踪哪些函数调用了其他函数以及哪些数据被修改了。为避免发生，结构化编程应运而生。

结构化编程的主要思想是分而治之。可将计算机程序看作由一系列任务组成。任何过于复杂、无法简单描述的任务都将分解为一系列较小的子任务，直至每个任务都很小，很容易理解。

例如，计算公司职员的平均工资是一项较复杂的任务。然而，可将其划分成下面几个子任务：

（1）确定公司员工数。

（2）确定每个员工的工资。

（3）计算工资总额。

（4）将工资总额除以员工数。

计算工资总额还可分成以下几步：

（1）读取每个职员的记录。

（2）读取工资额。

（3）将工资额添加到工资总额中。

（4）读取下一个职员的记录。

读取每个职员的记录又可分为以下几步：

（1）打开职员文件。

（2）找到正确的记录。

（3）读取数据。

结构化编程仍是一种非常成功的、处理复杂问题的方法。然而，到 20 世界 80 年代后期，它的许多不足逐渐暴露出来了。

首先，一种自然而然的愿望是，将数据（如职员记录）及其操作（排序、编辑等）看作一个整体。不幸

的是，结构化程序将数据结构与处理它们的函数分开，因此在结构化编程中，没有将数据同处理它的函数关联起来的自然方式。结构化编程通常被称为过程化编程，因为其着重于过程而不是对象。

其次，程序员们经常发现自己需要重用函数，但处理一种数据类型的函数通常不能用于处理其他类型的函数，这限制了重用函数带来的好处。

1.1.3 面向对象编程（OOP）

面向对象编程旨在满足以上种种需求，提供了一种管理极度复杂性的技术，实现了软件组件的可重用性，并将数据和操纵数据的任务结合起来。

面向对象编程的实质是模拟对象（东西或概念）而不是数据。要模拟的对象可以是屏幕上的小部件，如按钮、列表框，也可以是现实世界中的东西，如客户、自行车、飞机、猫和水。

对象有特征（属性），如快、宽敞、黑色、湿；也有功能，如购买、加速、飞、叫、冒泡。在编程语言中描述这些对象是面向对象编程的工作。

1.1.4 C++和面向对象编程

C++完全支持面向对象编程，它包括以下3个面向对象开发要素：封装、继承和多态。

1. 封装

工程师需要在正在制造的设备中添加电阻时，通常不会从头开始去制作电阻。他会到电阻库中，检查表示属性的彩色条纹，选择一个满足需求的电阻。对于这位工程师来说，电阻是一个黑盒子——只要电阻满足规格即可，他不必关心电阻的内部结构及其工作原理。

能够成为自包容单元的特性称为封装。利用封装可以实现数据隐藏。数据隐藏是一种非常有价值的特性，用户不必了解或关心对象的内部工作原理就可以使用它。就像使用冰箱时不必关心压缩机的工作原理一样，也可以使用设计优良的对象而不必了解其内部工作部件。内部工作部件的变更不会影响程序的运行，条件是它们满足指定的规格；就像可以更换冰箱中的压缩机一样。

同样，工程师在使用电阻时也不必知道电阻的内部状态。电阻的所有属性都封装在电阻对象中；它们不会通过电路传出。无需了解电阻的工作原理就可以有效地使用它。其内部工作部件隐藏在电阻壳内部。

C++通过创建被称为类的用户定义类型来支持封装。在第6章将介绍如何创建类。类一经创建，就是一个完全封装的实体——一个完整的单元。类的内部工作原理都可以隐藏。对于定义良好的类，用户只需知道如何使用它即可，而不必知道它是如何工作的。

2. 继承和重用

Acme 汽车公司的工程师们要制造一款新车时，他们有两种选择：从头做起或修改现有车型。也许他们的 Star 车型已接近完美，但他们想加一个涡轮增压器和一个六速变速装置。总工程师不想从头开始，于是说："我们另造一款 Star 车吧，但给它增加上述附加装置，并将把新型车叫做 Quasar。" Quasar 其实是 Star 车型的一种，只是增加了一些新特征的专用车而已。

C++支持继承。使用继承，可以声明一个新类型作为已有类型的扩展。新子类是从已有类型派生而来的，有时被称为派生类型。例如，如果 Quasar 由 Star 派生而来，它将继承了 Star 的所有品质，然后可以增加发动机或对其进行改进。继承及其在C++中的应用在第12章和第16章介绍。

3. 多态

当你踩下油门时，新的 Quasar 车型可能做出与 Star 车型不同的反应。Quasar 可能喷射燃料并启动涡轮增压器，而 Star 只是让汽油进入化油器中。但对用户而言，他不必知道这些差别。他只需踩下油门，根据他驾驶的是哪种车，正确的机器响应将随之发生。

C++支持这种想法，即通过函数多态和类多态，不同的对象将做正确的事情。多态是指同一名称有多种形式，将在第10章和第14章讨论。

1.2 C++的发展历程

当面向对象的分析、设计和编程开始兴起时，Bjarne Stroustrup 将最流行的商用软件开发语言 C 语言进行了扩展，使其可提供面向对象编程所需的功能。

尽管 C++是 C 的超集，且任何合法的 C 程序都是合法的 C++程序，但从 C 跳到 C++的意义是非常重大的。由于 C 程序员都能轻松地进入 C++世界，因此，多年来 C++从其与 C 的关系中获益匪浅。然而，想真正获得 C++带来的好处，很多程序员发现它们不得不放弃很多已有的观念，并学习一种新的分析和解决编程问题的方法。

1.3 应该先学习 C 语言吗

既然 C++是 C 的超集，那么学习 C++前是否该先学 C 呢？Stronstrup 和大多数 C++程序员都认为，不仅没有必要先学 C，而且不学可能比学了更好。

C 编程基于结构化编程概念；C++基于对象化编程。如果先学 C 语言，将不得不忘掉 C 语言提倡的坏习惯。

本书假定读者没有任何编程经验。不过，如果读者是一个 C 程序员，本书的前几章在很大程度上是一种复习。从第 6 章开始将开始介绍真正的面向对象软件开发。

1.4 C++、Java 和 C#

C++是在商业软件开发中占统治地位的语言之一。近年来，Java 一直在挑战这种统治地位，但现在很多转向 Java 的程序员又重新使用 C++。这两种语言在很多地方非常相似，因此学会了一种就学会了另一种的 90%。

C#是微软为其.NET 平台开发的新语言。C#使用的语法几乎与 C++相同，虽然它们在一些重要方面有所不同，但学会 C++后，便掌握了使用 C#所需的大部分知识。如果读者打算以后学习 C#，你在 C++上所做的努力将是不错的投资。

1.5 微软的 C++可控扩展

在.NET 中，微软公司新增了 C++可控扩展（Managed Extension），被称为 Managed C++。这种对 C++语言的扩展让 C++能够使用微软公司的新平台和函数库。更重要的是，Managed C++让 C++程序员能够充分利用.NET 环境的高级功能。要进行专门针对.NET 平台的开发，除了解有关标准 C++的知识外，还必须学习有关这些扩展的知识。

1.6 ANSI 标准

隶属于美国国家标准化协会的授权标准委员会（Accredited Standards Committee）制定了一个 C++国际标准。

该C++标准也被称为 ISO（国际标准化组织）标准、NCITS（国家信息技术标准化委员会）标准、X3（NCITS

的前身）标准和ANSI/ISO标准。由于ANSI标准使用最广，故本书沿用名称ANSI标准。

ANSI标准旨在确保C++是可移植的，例如，确保为Microsoft的编译器编写的符合ANSI标准的代码，用其他厂商的编译器进行编译时不会发生错误。由于本书的代码都符合ANSI标准，因此它们都可在Macintosh、Windows或Alpha计算机上通过编译。

对于大多数C++学习者，一般接触不到ANSI标准。最新的C++标准为ISO/IEC 14882-2003，前一版本已稳定使用了很长时间，所有的主要制造商都支持它。我们对本书的所有代码都同C++标准进行了比较，以确保它们符合该标准。

别忘了，并非所有的编译器都完全遵循C++标准。另外，在有些方面，C++标准没有做明确的规定，而是留给编译器厂商去选择，因此读者不能完全信任编译器，指望使用不同品牌的编译器进行编译时，程序的运行方式完全相同。

注意：由于C++可控扩展只适用于.NET平台，并非ANSI标准的组成部分，因此本书将不介绍这一主题。

1.7 编程准备

所有语言都要求编写程序前对程序作总体设计，C++对此要求更高。当然，本书前几章讨论的简单问题并不需要太多设计。然而，对于复杂问题，诸如那些专业程序员每天都会遇到的问题，必须进行设计。设计得越细致，在指定时间和预算内，设计出能够解决指定问题的程序的可能性越大。良好的设计常使程序的错误少且易于维护。据估计，软件成本的90%为调试和维护费用。良好的设计能够在很大程度上减少这些费用，对项目的最低成本有重要影响。

在准备设计任何程序之前，需要问的第一个问题是：要决什么问题？每个程序都应该有明确的目标，即便是本书中最简单的程序也如此。

非常优秀的程序员要问的第二个问题是：不自己编写软件能完成这些任务吗？重用旧程序、使用纸和笔或者购买现成的软件常常是比编写新程序更好的解决问题的方法。能够提供这种替代办法的程序员永远不愁找不到工作，为今天的问题找到廉价的解决方法总会给以后带来新的机会。

理解了要解决的问题，并确定需要为此编写一个新程序后，便可以开始设计了。

彻底搞清问题（分析）和制定解决方案（设计）是编写世界级商业应用程序所必需的基础。

1.8 开发环境

本书假设你的编译器有一种可以直接写入到控制台的模式（如MS-DOS命令提示符或外壳窗口），不用考虑诸如Windows和Macintosh这样的图形环境。请查找console或easy window这样的选项或查看编译器文档。

你的编译器可能是集成开发环境（IDE）的一部分，也可能有内置的源代码文本编辑器，或者你可能使用能生成文本文件的商用文本编辑器或字处理器。重要的是，无论在什么环境中编写程序，必须将其保存为简单的纯文本文件，文本中没有嵌入字处理命令。可安全使用的编辑器包括：Windows记事本、DOS Edit命令、Brief、Epsilon、Emacs和Vi。许多商用字处理软件如WordPerfect、Word等也都提供了保存简单文本文件的方法。

使用编辑器创建的文件称为源文件，对于C++，它们的扩展名通常为.cpp、.cp或.c。本书中所有源代码文件的扩展名都是.cpp，但请查看你的编译器，看它需要什么样的扩展名。

注意：大多数C++编译器并不在乎源代码文件的扩展名，但如果没有指明，许多编译器默认使用.cpp。然而，需要注意的是，有些编译器将.c文件视为C代码，而将.cpp文件视为C++代码。因此再次提醒你，请查看编译器文档。如果对C++源代码文件统一地使用扩展名.cpp，其他程序员将能够更容易地了解你的程序。

该做：请使用简单的文本编辑器创建源代码或编译器提供的内置编辑器。

保存文件时使用扩展名.c、.cp 或.cpp，推荐使用扩展名.cpp。

查看文档中有关编译器和链接器的信息，以确保你知道如何编译和链接程序。

不该做：不要使用保存特殊格式字符的字处理器。如果使用字处理器，必须将文件保存为 ASCII 文本文件。

如果编译器将扩展名为.c 的文件视为 C 代码而不是 C++代码，则不要使用这种扩展名。

1.9 创建程序的步骤

创建新程序的第一步是在源文件中编写合适的命令（语句）。尽管源文件中的源代码有点晦涩难懂，任何不懂 C++的人都很难理解，但它仍被称为人类可读的格式。源代码文件并不是程序，不能像程序那样执行或运行，但可执行程序文件可以。

1.9.1 用编译器生成对象文件

使用编译器可将源代码文件转换为程序。如何启动编译器以及如何告诉它在何处查找源代码随编译器而异；请查看编译器文档。

源代码编译后，将生成一个目标文件，其扩展名通常为.obj 或.o，然而它仍不是可执行程序。要把目标文件转换为可执行程序，必须运行链接器。

1.9.2 用链接器生成可执行文件

C++程序通常是将一个或多个.obj 文件与一个或多个库链接而生成的。库是编译器提供的一组可链接文件，也可能是单独购买或是程序员创建并编译而成的。所有 C++编译器都有一个包含有用函数和类的库，可以将其包含在程序中。接下来的几章将详细介绍函数和类。

创建可执行文件的步骤如下：

1. 创建扩展名为.cpp 的源代码。
2. 将源代码文件编译成扩展名为.obj 或.o 的目标文件。
3. 将目标文件与所需的各种库链接起来，生成可执行程序。

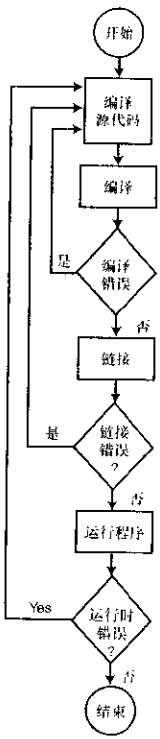

图 1.1 开发 C++程序的步骤

1.10 程序开发周期

如果每个程序都能在第一次运行时可行，则完整的程序开发周期为：编写程序、编译源代码、链接程序和运行。遗憾的是，几乎所有程序（无论它多简单）都会有一些错误。有些错误导致无法通过编译，另一些导致无法链接，还有一些仅在运行程序时才会表现出来（这些错误通常被称为 bug）。

无论发现什么类型的错误，都必须改正它，这涉及编辑源代码、重新编译、重新链接以及重新运行程序。图 1.1 描述了程序开发周期中的各个步骤。

1.11 HELLO.cpp：第一个 C++程序

传统的编程图书都以在屏幕上打印“Hello World”或其变体开始。本书秉承这个历史悠久的传统。

将程序清单 1.1 中的源代码原封不动（行号除外）地输入到编辑器中，确定输入无误后保存该文件，然后编译、链接并运行它。如果一切顺利，它将在屏幕上打印 Hello World。现在不必过多考虑该程序的工作原理，这里提供它旨在让读者熟悉开发周期。接下来的几章将详细介绍该程序的各个方面。

警告：在下面的程序清单中，左边给出了行号，供书中方便引用，不应将它们输入到编辑器中。例如，对于清单 1.1 的第 1 行，应该输入：

```
#include <iostream>
```

程序清单 1.1 HELLO.cpp

```
1:  #include <iostream>
2:
3:  int main()
4:  {
5:    std::cout << "Hello World!\n";
6:    return 0;
7:  }
```

请务必照原样输入，特别要注意标点符号。第 5 行中的<<是重定向符，在大多数键盘上可按下 Shift 键，再按两次逗号键输入。第 5 行中的 std 和 cout 之间是 2 个冒号（::）。第 5 行和第 6 行都以分号（;）结束。

另外，务必正确地使用编译器。大多数编译器都会自动链接，但请查看文档，看是否需要提供特殊的选项或执行命令来进行链接。

如果遇到错误，请仔细检查代码，看它与上面的程序清单有何不同。如果第 1 行有错，如 cannot find file iostream，请查看编译器文档，了解如何设置包含路径或环境变量。

如果出现指出没有 main 函数原型的错误，请在第 3 行前加一行：int main();（这是有些编译器使用的 main 函数原型）。在这种情况下，必须在本书中每个程序的 main 函数前添加上述代码行。大多数编译器都无此要求，只有少数编译器要求这样做。如果你的编译器是这样的，完成后的程序应如下：

```
#include <iostream>
int main();          // most compilers don't need this line
int main()
{
    std::cout <<"Hello World!\n";
    return 0;
}
```

注意：如果不知道特殊字符和关键字的发音，将难以阅读程序。第 1 行应读做“英镑符号 include eye-oh-stream”，第 5 行应读做“ess-tee-dee-see-out Hello World”。

在 Windows 系统上，试着运行 HELLO.exe（或在你的操作系统中相应的可执行文件。例如，在 UNIX 系统上，应运行 HELLO，在 UNIX 中可执行程序没有扩展名）。它将把 Hello World!直接写到屏幕上。如果情况果真如此，那么祝贺你！你输入、编译并运行了你的第一个 C++程序。它好像没什么了不起，但几乎每位专业 C++程序员都是从这个简单程序起步的。

使用 IDE（如 Visual Studio 或 Borland C++ Builder）的程序员可能发现，运行该程序时，弹出一个窗口，然后迅速消失，根根没机会看清程序的输出。在这种情况下，请在源代码的 return 语句之前添加下述代码行：

```
char response;
std::cin >> response;
```

这些代码行导致程序暂停，直到用户输入一个字符（可能还需要按下回车键）。它们让用户有机会看到运行结果。如果对于程序 hello.cpp 需要这样做，则对于本书中的大部分程序也需要这样做。

使用标准库

如果使用的是非常旧的编译器，上述程序将无法运行——找不到新的 ANSI 标准库。在这种情况下，请按如下修改该程序：

```
1:  #include <iostream.h>
2:
3:  int main()
4:  {
5:      cout << "Hello World!\n";
6:      return 0;
7:  }
```

注意，现在库名以.h 结尾，在第 5 行不再在 cout 前使用 std::。这是 ANSI 标准之前的老式头文件样式。如果你的编译器采用这种方式，而不是前面的方式，则是过时的。在本书前几章，该编译器还可行，但涉及模板和异常时，你的编译器就不管用了。

1.12 编译器初步

本书不针对特定的编译器。这意味着本书的程序可以在任何平台（Windows、Mac、UNIX、Linux 等）上使用任何遵循 ANSI 标准的 C++编译器进行编译。

当前，大多数程序员在 Windows 环境下编程，大多数专业程序员使用 Microsoft 编译器。这里无法介绍每种编译器的编译和链接细节，但将介绍如何使用 Visual C++ 6，这种编译器的用法与其他编译器相似。

然而，由于编译器的差异，请务必查看编译器文档。

创建 Hello World 项目

要创建并测试 Hello World 程序，请按以下步骤执行：

（1）打开编译器。

（2）从菜单中选择 File/New。

（3）选择 Win32 Console Application，然后输入项目名，如 hello，最后单击 OK 按钮。

（4）从选项菜单中选择 An Empty Project，单击 Finish 按钮。将出现一个对话框，其中包含新项目的信息。

（5）单击 OK 按钮，回到主编辑器窗口。

（6）从菜单中选择 File/New。

（7）选择 C++ Source File，并在文本框 File Name 中输入一个名称，如 hello。

（8）单击 OK 按钮，回到主编辑器窗口。

（9）输入前面所示的代码。

（10）选择菜单 Build/Build hello.exe。

（11）确保没有编译错误，这种信息出现在编辑器底部附近。

（12）按 Ctrl+F5 或选择菜单 Build/Execute hello 来运行该程序。

（13）按空格键结束程序。

FAQ

我能够运行该程序，但窗口稍瞬即失，无法阅读运行结果。请问哪里出现了问题？

答：查看编译器文档，应该有让程序暂停的方法。对于 Microsoft 编译器，方法是按 Ctrl + F5。

对于其他编译器，可以在返回语句之前（即程序清单 1.1 的第 5 和第 6 行之间）添加如下代码：

```
char response;
std::cin >> response;
```

这将导致程序暂停，等待用户输入一个字符。要结束程序，可输入任何字母或数字（如 1），然后按回车键（如果必要的话）。

std::cout 和 std::cin 的含义将在接下来的几章中讨论，现在只需将其作为神奇的咒语使用即可。

1.13 编译错误

发生编译错误的原因有很多，通常是由于输入错误或其他不经意的小错误引起的。优秀的编译器不仅能告诉你发生了什么错误，还会准确指出代码中的出错位置。更好一些的编译器甚至能指出修改方法。

可以故意在程序中加入一个错误来验证这一点。如果 HELLO.cpp 运行正常，请编辑它并把程序清单 1.1 中第 7 行的括号去掉，修改后的程序如程序清单 1.2 所示。

程序清单 1.2 演示编译错误

```
1: #include <iostream>
2:
3: int main()
4: {
5:   std::cout << "Hello World!\n";
6:   return 0;
```

重新编译该程序，你将看到类似下面这样的错误消息：

```
Hello.cpp(7) : fatal error C1004: unexpected end of file found
```

错误消息指出有问题的文件和行号以及是什么问题（尽管有点难懂）。在这里，错误消息指出，找遍整个源代码文件，直到到达末尾也没有找到右大括号。

有时候，错误消息只指出错误的大概位置。如果编译器能够准确地识别每个问题，将能够修复代码。

1.14 小 结

阅读本章后，读者应该对 C++的发展历程及其用于解决什么问题有深入的认识。读者应该深信，对任何对编程感兴趣的人来说，学习 C++都是正确的选择。C++提供了面向对象编程工具及系统级语言性能，这使其成为当今开发语言的最佳选择。

今天你学习了如何输入、编译、链接和运行第一个 C++程序，并了解了程序开发周期。你还对什么是面向对象编程有一定的了解。在接下来的三周中，你还将不断学习这些主题。

1.15 问 与 答

问：文本编辑器和字处理器之间有何区别？

答：文本编辑器生成的文件是纯文本文件，没有字处理器所需的格式化命令或特殊符号。简单文本编辑器不支持自动换行、粗体、斜体等。

问：如果编译器有内置的编辑器，必须使用它吗？

答：几乎所有的编译器都能编译由任何文本编辑器生成的代码。然而，使用内置文本编辑器的好处是，在开发周期的编辑和编译步骤之间可以快速地来回移动。高级编译器包含一个完全集成的开发环境，让程序员能够访问帮助文件、就地编辑和编译代码，同时不必离开开发环境即可解决编译和链接错误。

问：可以忽略编译器发出的警告消息吗？

答：编译器通常会指出错误和警告。如果有错误，程序将不能通过编译。如果只有警告，编译器通常是创建可执行程序。

很多书籍对这个问题都含糊其辞。正确的答案是不能！一开始就应养成将警告消息视为错误的习惯。当你所做的事并非你本意时，C++将通过编译器向你发出警告。请认真对待这些警告并设法消除它们。有些编译器甚至一个设置选项，将所有警告视为错误，进而禁止生成可执行程序。

问：什么是编译阶段？

答：编译阶段是指运行编译器的阶段，与链接阶段（运行链接器时）和运行阶段（运行程序时）相对。这只是程序员用来表示错误通常出现的一种阶段的简称。

1.16 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。

1.16.1 测验

1. 解释器和编译器有何不同？
2. 如何使用编译器编译源代码？
3. 链接器的作用是什么？
4. 开发周期包括哪些步骤？

1.16.2 练习

1. 阅读下面的程序，不运行它的情况下猜测其功能：

```
1: #include <iostream>
2: int main()
3: {
4:    int x = 5;
5:    int y = 7;
6:   std::cout << endl;
7:   std::cout << x + y << " " << x * y;
8:   std::cout << end;
9:   return 0;
10:}
```

2. 输入练习 1 中的程序，然后编译并链接它。它做什么？与你的猜测相符吗？
3. 输入并编译下面的程序。发生了什么错误？

```
1: include <iostream>
2: int main()
3: {
4:    std::cout << "Hello World \n";
5:    return 0;
6: }
```

4. 修复练习 3 中程序的错误，重新编译、链接并运行它。它做什么？