C++17 入门经典

(第 5 版)

[美] 艾佛 • 霍尔顿(Ivor Horton)
彼得 • 范维尔特(Peter Van Weert) 著
卢旭红　张骏温 译

清华大学出版社
北　京

Beginning C++17, 5th Edition
Ivor Horton, Peter Van Weert
EISBN：978-1-4842-3365-8
Original English language edition published by Apress Media. Copyright © 2018 by Apress Media. Simplified Chinese-Language edition copyright © 2019 by Tsinghua University Press. All rights reserved.

本书中文简体字版由 Apress 出版公司授权清华大学出版社出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号　图字：01-2018-5181

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

C++17 入门经典：第 5 版/ (美)艾佛·霍尔顿(Ivor Horton)，(美)彼得·范维尔特(Peter Van Weert) 著；卢旭红，张骏温 译. —北京：清华大学出版社，2019

书名原文：Beginning C++17, 5th Edition
ISBN 978-7-302-52769-5

Ⅰ. ①C… Ⅱ. ①艾… ②彼… ③卢… ④张… Ⅲ. ①C++语言－程序设计 Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字(2019)第 071023 号

责任编辑：王　军　李维杰
装帧设计：孔祥峰
责任校对：成凤进
责任印制：丛怀宇

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市铭诚印务有限公司
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**32.75　　**字　　数：**1205 千字
版　　次：2019 年 6 月第 1 版　　**印　　次：**2019 年 6 月第 1 次印刷
定　　价：99.00 元

产品编号：081003-01

译 者 序

第一次接触 C++应该是在 20 世纪 90 年代初，一晃已经快 30 年了。C++作为编程行业资深的计算机语言，自 20 世纪 80 年代初被发明以来，迭代速度一直比较慢，2010 年之前只更新过两个版本，2011 年后焕发青春，每三年更新一次，2017 年新公布的 ISO/IEC 14882:2017 C++17 是目前最新的版本。这次能荣幸地参与《C++17 入门经典(第 5 版)》的翻译工作感到十分荣幸。

作者 Ivor Horton 是著名的计算机语言教材作家，他在工作中精通各种计算机语言，出版了大量的编程教材，涉及 C、Java、VC++和 C++等。此书由 Ivor Horton 和 Peter Van Weert 在 2018 年合作完成，是一本优秀的基于 C++17 标准的初学者指南。

《C++17 入门经典(第 5 版)》不仅仅针对 C++17 新增的特性，而且从计算机中最基础的数字和字符的表示形式开始从整体上介绍 C++编程知识，读者可以对 C++语言有个全面的了解，编写简单的 C++程序。这本书通俗易懂，作者通过大量案例介绍每个概念和语言特性，每章最后还布置了作业，让读者可以对自己所学的内容进行复习和练习。本书适合于大多数开发环境，可以在常见的操作系统或程序开发系统中使用，读者不需要具备任何编程基础。

C++是一种面向对象编程语言，用它编写的程序优美、可读性好，生成的代码质量和运行效率都较高，已经成为当今主流程序设计语言中对程序员要求最高的一种语言。要想学好 C++这种计算机语言，单纯读一本书应该是远远不够的，重要的是动手写程序，然后对所写的程序进行编译和运行，并且需要进一步调试程序和分析结果。学习计算机语言最重要的功课应该是：练习、练习、再练习。

感谢清华大学出版社各位编辑提供的大力支持，他们花了大量时间审核和编辑此书，做了大量的基础工作。本书在翻译过程中力求尊重原文，某些地方的译法可能与目前的常用说法稍有不同。另外本书页数较多，虽然检查了几遍，难免还会有一些疏漏，敬请大家谅解。

作者简介

Ivor Horton 从数学系毕业，却被信息技术领域工作量少、回报高的前景所吸引。虽然现实证明，工作量大，回报相对一般，但是他与计算机一直相伴到今天。在不同的时期，他参与过编程、系统设计、咨询以及相当复杂的项目的管理和实施工作。

Ivor 有多年工程设计和制造控制系统的设计和实施经验。他使用多种编程语言开发过在不同场景中很实用的应用程序，并教会一些科学家和工程师如何使用编程语言开发一些实用的程序。他目前出版的图书涉及 C、C++和 Java 等编程语言。当他没有在撰写编程图书或者为他人提供咨询服务时，他会去钓鱼或旅行，享受生活。

Peter Van Weert 是一名软件工程师，主要兴趣和专长是应用软件开发、编程语言、算法和数据结构。他在鲁汶大学以最优毕业生荣誉获得计算机科学硕士学位，并得到了考试委员会的祝贺。2010 年，他在鲁汶大学的声明式编程语言和人工智能研究组完成了博士论文，主题是基于规则的编程语言的设计和高效编译。在攻读博士期间，他担任面向对象编程(Java)、软件分析与设计以及声明式编程的助教。

毕业后，Peter 在 Nikon Metrology 工作了 6 年多，负责 3D 激光扫描和点云检查领域的大规模工业应用软件设计。他学习并精通 C++以及极大规模代码库的重构和调试，并进一步熟悉了软件开发过程的各个方面，包括功能和技术需求的分析，以及敏捷的、基于 Scrum 的项目和团队管理。

如今，Peter 就职于 Danaher 的数字牙医软件研发部，为未来的牙医业开发软件。

在空余时间，他与人合作撰写了两本关于 C++的图书，开发了两个获奖的 Windows 8 应用，并且是比利时 C++用户组的定期专家演讲人和董事会成员。

技术审校者简介

Marc Gregoire 是一名软件工程师。他毕业于比利时鲁汶大学，拥有“Burgerlijk ingenieur in de computer wetenschappen”学位(相当于计算机工程硕士)。毕业后的第二年，他在鲁汶大学以优秀毕业生荣誉获得人工智能硕士学位。离开学校后，Marc 就职于一家软件咨询公司 Ordina Belgium。作为咨询顾问，他服务于 Siemens 和 Nokia Siemens Networks 公司，为电信运营商开发运行在 Solaris 系统上的关键的 2G 和 3G 通迅软件。Marc 具有国际化团队合作开发经验，与来自南美、欧洲、中东以及亚洲的团队成员协同工作，也曾与来自美国的团队合作。目前，Marc 就职于 Nikon Metrology，负责开发工业 3D 激光扫描软件。

前　言

欢迎阅读本书。本书是 Ivor Horton 撰写的 *Beginning ANSI C++*的修订更新版本。自从那本书出版以后，C++语言已经被大量扩展和改进，使得如今已经无法在一本书中详细解释完整的 C++语言。本书将介绍 C++语言和标准库特性的基本知识，足以让读者开始编写自己的 C++程序。学习完本书后，读者将有能力扩展自己的 C++技能的深度和广度。

我们假定读者没有任何编程经验。如果读者乐于学习，并且擅长逻辑思维，那么理解 C++并没有想象中那么难。通过开发 C++技能，读者将学习一种已经有数千万人使用的编程语言，而且这种语言能够用来在几乎任何环境中开发应用程序。

C++非常强大，甚至可以说，它比大部分编程语言更加强大。所以，就像任何强大的工具一样，如果不经训练就开始使用，可能造成严重伤害。我们常把 C++比作瑞士军刀：由来已久，大众信任，极为灵活，但也可能令人茫然，并且到处是尖锐的东西，可能伤害自己。但是，当有人明明白白解释不同工具的用途，并讲解一些基本的用刀安全守则之后，就再也不需要寻找其他小型刀具了。

学习 C++并不像想象中的那样具有很大危险或困难。如今的 C++要比许多人想象中的更容易理解。自从 40 年前 C++语言问世之后，已经有了长足的改进。本质上，我们已经学会了如何以最安全有效的方式来使用其强大的刀刃和工具。而且，可能更重要的是，C++语言及其标准库也相应地发生了演化，更便于使用。特别是，过去十年间，“现代 C++”开始崛起。现代 C++强调使用更新、更具表达力、更安全的语言特性，并结合经过实践验证的最佳实践和编码指导原则。当知道并应用一些简单的规则和技术后，C++的许多复杂性将随之消失。关键在于有人能够不只恰当地、循序渐进地解释 C++能做什么，还能解释应该怎么使用 C++去做。这正是本书的目的。

在这本最新修订版中，我们不遗余力，使内容跟上 C++编程的这个新时代。当然，与以前的版本一样，我们仍然采用轻松的、循序渐进的方式进行讲解。我们使用许多实用的编码示例和练习题，展示 C++旧有的和新增的所有刀刃。还不只如此：我们更加努力地确保总是解释实现某种目的的最适合工具，为什么如此选择，以及如何避免造成失误。我们确保读者从一开始学习 C++，就采用安全高效的现代编程风格，而这会是未来的雇主们希望员工具备的技能。

本书讲解的 C++语言对应于最新的国际标准化组织(International Organization for Standardization，ISO)标准，常被称为 C++17。但是，我们并没有介绍 C++17 的全部内容，因为相比 C++语言的之前版本，C++17 所做的许多扩展针对的是高级应用。本书的所有示例均可使用支持 C++17 的编译器编译执行。

如何使用本书

要通过本书学习 C++，需要有一个支持 C++17 标准的编译器和一个适合编写程序代码的文本编辑器。目前有一些编译器支持 C++17，其中有几个是免费的。

GCC 和 Clang 编译器对 C++17 提供了全面支持，并且二者都是开源的，可免费下载。对于新手，安装这两个编译器，并将其与合适的编辑器关联起来，并不容易。安装 GCC 和合适的编辑器，有一种简单的方法，即下载 Code::Blocks 或 Qt Creator。它们都是免费的集成开发环境(Integrated Development Environment，IDE)，可用于 Linux、Apple macOS 和 Microsoft Windows。它们支持使用几种编译器进行完整的程序开发，其中包括 GCC 和 Clang。这意味着安装了它们可同时得到对 C 和 C++的支持。

另一种选择是使用 Microsoft Visual C++，它运行在 Microsoft Windows 上。Microsoft Visual C++几乎完全支持

C++17；本书的所有示例应该能够在最新版本的 Microsoft Visual C++上正确编译。其 Community 版本和 Express 版本可供个人甚至小规模专业团队免费使用。Visual Studio 提供了一个功能全面的专业编辑器，以及对其他语言(如 C#和 Basic)的支持。

还有其他一些编译器也支持 C++17，在网上进行搜索可了解它们。本书提供的下载文件中还包含一个清单，其中列出了其他一些可帮助入门的有用资源(读者可通过扫描封底二维码来下载本书源代码)。

本书内容应当按顺序阅读，所以读者应该从头读起，直到读完本书。但是，只通过读书是无法学会编程的。只有实际编写代码，才能学会如何用 C++编写程序，所以一定要自己键入所有示例，而不要简单地从下载文件中复制代码，然后编译并执行自己键入的代码。这项工作有时候看起来会很枯燥，但是读者会惊奇地发现，仅仅键入 C++语句就能够对理解 C++有巨大帮助，尤其是感觉难以理解某些思想的时候更是如此。如果某个示例不能工作，先不要急于翻看本书来查找原因。试着从代码中分析什么地方出错。这是一种很好的练习，因为在实际开发 C++应用程序时，更多的时候需要自己分析代码。

犯错是学习过程中不可缺少的一部分，书中的练习题给了读者大量机会来犯错。自己设计一些练习题是一个好主意。如果不确定怎么解决一个问题，先自己试一试，然后查看答案。犯错越多，对什么地方会出错的理解就越深刻。确保完成所有练习题，并且要记住，只有确定自己解决不了问题时才查看答案。大部分练习题只需要直接运用对应章节中的知识，换言之，它们只是练习而已，但是也有一些练习题需要深入思考，甚至需要一些灵感。

我们希望读者能够掌握 C++，并且最重要的是，享受使用 C++编写程序的过程。

Ivor Horton
Peter Van Weert

目　录

第 1 章

基 本 概 念

为了让读者更好地理解书中的案例，本章将概述 C++的主要元素及其组合方式，以帮助读者理解这些元素，并探讨计算机中数字和字符表达的几个概念。

本章主要内容

- 现代 C++的含义
- 术语 C++11、C++14 和 C++17 的含义
- C++标准库
- C++程序的元素
- 如何注释程序代码
- C++代码如何变成可执行程序
- 面向对象编程与面向过程编程的区别
- 二进制、十六进制和八进制数字系统
- 浮点数
- 计算机如何只使用位和字节来表示数字
- Unicode

1.1 现代 C++

C++编程语言最初是由丹麦计算机科学家 Bjarne Stroustrup 在 20 世纪 80 年代初发明的，是至今仍然被广泛使用的“古老的”编程语言之一。事实上，在计算机编程这个节奏快速的世界中，可以算是非常古老了。然而，尽管已有多年的历史，C++仍然表现强势，在最有名的编程语言受欢迎度排行榜中，稳居前 5 名。毫无疑问，C++仍然是目前世界上使用最广泛、最强大的编程语言之一。

几乎任何程序都可以用 C++编写：设备驱动程序、操作系统、薪酬管理程序、游戏等。随意说出来一个主流的操作系统、浏览器、办公套件、电子邮件客户端、多媒体播放器或数据库系统，它们有很大的可能至少用 C++写了一部分功能。最重要的是，C++可能最适合用来编写对性能有较高要求的应用程序，例如需要处理大量数据的应用程序、具有复杂图形处理的现代游戏、针对嵌入式设备或移动设备的应用程序。使用 C++编写的程序仍然比使用其他流行语言编写的程序快许多倍。而且，对于在多种多样的计算设备和环境中开发应用程序，包括个人电脑、工作站、大型计算机、平板电脑和移动电话，C++远比其他大部分语言高效得多。

虽然 C++编程语言已经不年轻，但是依然充满活力，近年来显得更加生机勃勃。在 20 世纪 80 年代，C++被发明出来并标准化以后，演化速度并不快，直到 2011 年，国际标准化组织(International Organization for Standardization，ISO)发布 C++编程语言标准的一个新的版本。该版本常被称为 C++11，它使 C++得到复苏，将这个稍许有些过时的语言直接带入 21 世纪。C++11 深刻地现代化了 C++语言和我们使用这种语言的方式，以至于我们几乎可以将 C++11 看成一种新的语言。

使用 C++11 及更新版本的功能进行编程称为“现代 C++”编程。本书将展示，现代 C++并不只是简单地拥抱 C++语言的新功能，如 lambda 表达式、自动类型推断和基于范围的 for 循环等。最重要的是，现代 C++代表现代的

编程方法，是人们对良好编程风格达成的共识。它运用一套指导原则和最佳实践，使C++编程更加简单、更难出错并且生产效率更高。C++11 是一种现代的、安全的编程风格，将传统的低级语言结构替换为容器(第 5 章和第 19 章)、智能指针(第 6 章)或其他 RAII 技术(第 15 章)，并且强调了使用异常来报告错误(第 15 章)、通过移动语义来按值传送对象(第 17 章)，以及编写算法而不是循环(第 19 章)等。当然，现在读者可能还不太理解它们是什么。但不必担心，本书将循序渐进地介绍使用现代 C++进行编程所需要的全部知识！

C++标准也使得 C++社区再次活跃起来，该标准问世之后，C++社区就一直在努力地扩展和进一步改进这种语言。每 3 年，就会发布该标准的一个新版本。2014 年，C++14 标准发布；2017 年，C++17 标准发布。本书讲解的是 C++17 标准定义的 C++。本书提供的所有代码，在支持 C++17 标准的任何编译器上均可以工作。好消息是，大部分主流编译器都紧跟最新的 C++发展，所以如果读者使用的编译器现在还不支持某个特定的功能，很快就会支持。

1.2 标准库

如果每次编写程序时，都从头开始创建所有内容，就是一件非常枯燥的工作。许多程序都需要相同的功能，例如从键盘上读取数据、计算平方根、将数据记录按特定的顺序排列。C++附带大量预先编写好的代码，提供了所有这些功能，因此不需要自己编写它们。这些标准代码都在标准库中定义。

C++带有一个非常大的标准库，其中包含大量例程和定义，提供了许多程序需要的功能。例如，数值计算、字符串处理、排序和搜索、数据的组织和管理、输入输出等。本书的几乎每一章都将介绍标准库的一些主要功能，并且在第 19 章，将更详细地介绍一些关键的数据结构和算法。尽管如此，由于标准库非常大，本书仅涉及皮毛。详细描述标准库提供的所有功能，需要好几本书的篇幅。*Beginning STL*(Apress，2015)是使用标准模板库的优秀指南，而标准模板库是 C++标准库中以各种方式管理和处理数据的一个子集。我们还推荐阅读 *C++ Standard Library Quick Reference*(Apress，2016)，这本书提供了对现代标准库的全面概览。

就 C++语言的范围和库的广度而言，初学者常常觉得 C++令人生畏。将 C++的全部内容放在一本书里是不可能的，但其实不需要学会 C++的所有内容，就可以编写实用的程序。该语言可以循序渐进地学习，这并不是很难。例如学习开车，即使没有赛车所需要的专业技能、知识和经验，也可以成为合格的、安全驾驶的司机。通过本书，可以学到使用 C++高效编程所需要的所有知识。读完本书后，你可以自信地编写自己的应用程序，还可以开始研究 C++及其标准库的所有内容。

1.3 C++程序概念

本书后面将详细论述本节介绍的所有内容。图 1-1 展示了一个完全可以工作的完整 C++程序，并解释了该程序的各个部分。这个示例将用作讨论 C++一般内容的基础。

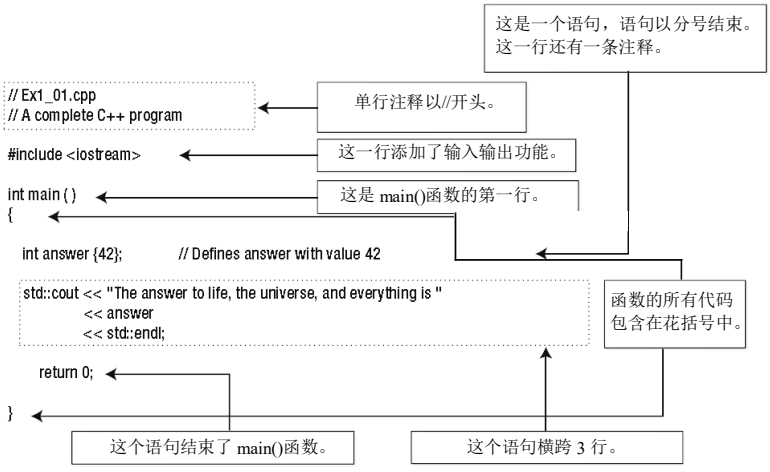

图 1-1　一个完整的 C++程序

1.3.1 源文件和头文件

图 1-1 中显示的文件 Ex1_01.cpp 包含在本书的源代码中。文件扩展名.cpp 表示这是一个 C++源文件。源文件包含函数和全部可执行代码。源文件的名称通常带有扩展名.cpp，不过有时候也使用其他扩展名来标识 C++源文件，例如.cc、.cxx 或.c++。

C++代码实际上保存在两种文件中。除了源文件，还有所谓的头文件。头文件包含许多内容，其中包括.cpp 文件中的可执行代码使用的函数原型，以及使用的类和模板的定义。头文件的名称通常带有扩展名.h，不过也可以使用.hpp。第 10 章将创建我们的第一个头文件。第 10 章之前的程序都很小，在一个源文件中定义就足够了。

1.3.2 注释和空白

图 1-1 中的前两行是注释。添加注释来解释程序代码，可以使他人更容易理解程序的逻辑。编译器会忽略一行代码中双斜杠后面的所有内容，所以这种注释可以跟在一行代码的后面。第一行注释指出包含代码的文件名。本书每个可工作示例的文件都以这种方式给出。

注意：

在每个头文件或源文件中，用注释指出文件名只是为读者提供方便。在日常编码中，没必要添加这种注释；如果添加了这种注释，只会在重命名文件时，引入不必要的维护开销。

需要把注释放在多行时，可以使用另一种注释形式。例如：

```
/* This comment is
   over two lines. */
```

编译器会忽略/*和*/之间的所有内容。可以修饰这种注释，使其更加突出。例如：

```
/*******************\
 * This comment is *
 * over two lines. *
\*******************/
```

空白是空格、制表符、换行符和换页符的任意序列。编译器一般会忽略空白，除非由于语法原因需要使用空白把元素区分开来。

1.3.3 预处理指令和标准库头文件

图 1-1 中的第三行是一个预处理指令。预处理指令会以某种方式修改源代码，之后把它们编译为可执行的形式。这个预处理指令把 iostream 标准库头文件的内容添加到这个源文件 Ex1_01.cpp 中。头文件的内容将被插入到#include 指令的位置。

头文件包含源文件中使用的定义。iostream 包含使用标准库例程从键盘输入以及将文本输出到屏幕上所需的定义。具体而言，它定义了 std::cout 和 std::endl。如果在 Ex1_01.cpp 中省略包含 iostream 头文件的预处理指令，源文件就不会编译，因为编译器不知道 std::cout 和 std::endl 是什么。在编译之前，把头文件的内容包含到源文件中。每个程序都可以包含一个或多个标准库头文件的内容，我们也将创建和使用自己的头文件，以包含本书后面创建的定义。

警告：

尖括号和标准头文件名之间没有空格。对于一些编译器而言，尖括号<和>之间的空格很重要；如果在这里插入空格，程序将不会编译。

1.3.4 函数

每个 C++程序都至少包含一个以上的函数。函数是一个命名的代码块，执行定义好的操作，例如读取输入的数据、计算平均值或者输出结果。在程序中使用函数的名称执行或调用函数。程序中的所有可执行代码都放在函数中。

程序中必须有一个名为 main 的函数，执行总是自动从这个函数开始。main()函数通常调用其他函数，这些函数又可以调用其他函数，以此类推。函数提供了几个重要的优点：

- 程序被分解为不同单元的函数，更容易开发和测试。
- 一个函数可以在程序的几个不同的地方重用，避免了在与每个需要的地方编写相同代码相比，这会使程序更小。
- 函数常常可以在许多不同的程序中重用，节省了编码时间和精力。
- 大程序常常由一组程序员共同开发。每个组员负责编写一系列函数，这些函数是整个程序中已定义好的一个子集。没有函数结构，这是不可能实现的。

图 1-1 中的程序只包含 main()函数。该函数的第一行是：

```
int main()
```

这称为函数头，标识了函数。其中 int 是一个类型名称，它定义了 main()函数执行完毕时返回的值的类型为整型。整数是没有小数部分的数字。例如，23 和 －2048 是整数，但 3.1415 和 1/4 不是。一般情况下，函数定义中名称后面的圆括号，包含了调用函数时要传递给函数的信息的说明。本例中的圆括号是空的，但其中可以有内容。第 8 章将学习如何指定执行函数时传递给函数的信息的类型。本书正文中总是在函数名的后面加上圆括号，例如 main()，以区分函数与其他代码。

函数的可执行代码总是放在花括号中，左花括号跟在函数头的后面。

1.3.5 语句

语句是 C++程序的基本单元。语句总是以分号结束。分号表示语句的结束，而不是代码行的结束。语句可以定义某个元素，例如计算或者要执行的操作。程序执行的所有操作都是用语句指定的。语句按顺序执行，除非有某个语句改变了这个顺序。第 4 章将学习可以改变执行顺序的语句。图 1-1 所示的 main()函数中有 3 个语句。第一个语句定义了一个变量，变量是一个命名的内存块，用于存储某种数据。在本例中变量的名称是 answer，可以存储整数值：

```
int answer {42};             // Defines answer with the value 42
```

类型 int 放在名称的前面，这指定了可以存储的数据类型——整数。注意 int 和 answer 之间的空格。这里的一个或多个空白字符是必需的，用于分隔类型名称和变量名称。如果没有空格，编译器会把名称看作 intanswer，这是编译器无法理解的。answer 的初始值放在变量名后面的花括号中，所以它最初存储了 42。answer 和{42}之间也有一个空格，但这个空格不是必需的。下面的所有定义都是有效的：

```
int one{ 1 };
int two{2};
int three{
   3
};
```

大多数时候，编译器会忽略多余的空格。但是，应以统一的风格使用空白，以提高代码的可读性。

在第一个语句的末尾有一个多余的注释，它解释了上述内容，这还说明，可以在语句中添加注释。//前面的空白也不是强制的，但最好保留这些空白。

可以把几个语句放在一对花括号{}中，此时这些语句就称为语句块。函数体就是一个语句块，如图 1-1 所示，main()函数体中的语句就放在花括号中。语句块也称为复合语句，因为在许多情况下，语句块可以看作一个语句，详见第 4 章中的决策功能，以及第 5 章中的循环功能。在可以放置一个语句的任何地方，都可以放置一个包含在花括号对中的语句块。因此，语句块可以放在其他语句块内部，这个概念称为嵌套。事实上，语句块可以嵌套任意级。

1.3.6 数据的输入输出

在 C++中，输入和输出是使用流来执行的。如果要输出消息，可以把消息写入输出流中；如果要输入数据，则从输入流读取。因此，流是数据源或数据接收器的一种抽象表示。在程序执行时，每个流都关联着某台设备，关联着数据源的流就是输入流，关联着数据目的地的流就是输出流。对数据源或数据接收器使用抽象表示的优点是，无

论流代表什么设备，编程都是相同的。例如，从磁盘文件中读取数据的方式与从键盘上读取完全相同。在C++中，标准的输出流和输入流称为 cout 和 cin，默认情况下，它们分别对应计算机的屏幕和键盘。第 2 章将从 cin 中读取输入。

在图 1-1 中，main()函数中的下一个语句把文本输出到屏幕：

```
std::cout << "The answer to life, the universe, and everything is "
          << answer
          << std::endl;
```

把该语句放在 3 行上，只是为了说明这么做是可行的。名称 cout 和 endl 在 iostream 头文件中定义。本章后面将解释 std::前缀。<<是插入运算符，用于把数据传递到流中。第 2 章会遇到提取运算符>>，它用于从流中读取数据。每个<<右边的所有内容都会传递到 cout 中。把 endl 写入 std::cout，会在流中写入一个换行符，并刷新输出缓存。刷新输出缓存可确保输出立即显示出来。该语句的执行结果如下：

```
The answer to life, the universe, and everything is 42
```

可以给每行语句添加注释。例如：

```
std::cout << "The answer to life, the universe, and everything is " // This statement
          << answer                                                 // occupies
          << std::endl;                                             // three lines
```

双斜杠不必对齐，但我们常常对齐双斜杠，使之看起来更整齐，代码更容易阅读。当然，不应该只是为了写注释而写注释。注释通常应该包含在代码中无法明显看出来的有用信息。

1.3.7 return 语句

main()函数中的最后一个语句是 return。return 语句会结束函数，把控制权返回给调用函数的地方。在本例中它会结束函数，把控制权返回给操作系统。return 语句可能返回一个值或没有返回值。本例的 return 语句给操作系统返回 0，表示程序正常结束。程序可以返回非 0 值，例如 1、2 等，表示不同的异常结束条件。Ex1_01.cpp 中的 return 语句是可选的，可以忽略它。这是因为如果程序执行超过了 main()函数中的最后一个语句，就等价于执行 return 0。

注意：

只有在 main()函数中，忽略 return 才相当于返回 0。对于其他任何返回类型为 int 的函数，最好以一个显式的 return 语句结束，否则编译器不知道任意函数在默认情况下应该返回哪个值。

1.3.8 名称空间

大项目会同时涉及几个程序员。这可能会带来名称问题。不同的程序员可能给不同的元素使用相同的名称，这可能带来一些混乱，使程序出错。标准库定义了许多名称，很难全部记住。不小心使用了标准库名称也会出问题。名称空间就是用于解决这个问题的。

名称空间类似于姓氏，置于该名称空间中声明的所有名称的前面。标准库中的名称都在 std 名称空间中定义，cout 和 endl 是标准库中的名称，所以其全名是 std::cout 和 std::endl。其中的两个冒号有一个非常奇特的名称：作用域解析运算符，详见后面的说明。这里它用于分隔名称空间的名称 std 和标准库中的名称(例如 cout 和 endl)。标准库中的几乎所有名称都有前缀 std。

名称空间的代码如下所示：

```
namespace my_space {

   // All names declared in here need to be prefixed
   // with my_space when they are reference from outside.
   // For example, a min() function defined in here
   // would be referred to outside this namespace as my_space::min()

}
```

花括号对中的所有内容都位于 my_space 名称空间中。第 10 章将详细介绍如何定义自己的名称空间。

警告：

main()函数不能定义在名称空间中，未在名称空间中定义的内容都存在于全局名称空间中，全局名称空间没有名称。

1.3.9 名称和关键字

Ex1_01.cpp 包含变量 answer 的定义，并使用在 iostream 标准库头文件中定义的名称 cout 和 endl。程序中的许多元素都需要名称，定义名称的规则如下：

- 名称可以是包含大小写拉丁字母 A～Z 和 a～z、数字 0～9 和下划线_的任意序列。
- 名称必须以字母或下划线开头。
- 名称是区分大小写的。

C++标准允许名称有任意长度，但有的编译器对此有某种长度限制，这个限制常常比较宽松，并不严格。大多数情况下，不需要使用长度超过 12~15 个字符的名称。

下面是一些有效的 C++名称：

```
toe_count shoeSize Box democrat Democrat number1 x2 y2 pValue out_of_range
```

大小写字母是有区别的，所以 democrat 和 Democrat 是不同的名称。编写由两个或多个单词组成的名称时，有几个约定；可以把第二个及以后各个单词的首字母大写，或者用下划线分隔它们。

关键字是 C++中有特殊含义的保留字，不能把它们用于其他目的。例如，class、double、throw 和 catch 都是保留字。其他不应使用的名称包括：

- 以连续两个下划线开头的名称。
- 以一个下划线后跟一个大写字母开头的名称。
- 在全局名称空间内所有以下划线开头的名称。

虽然使用了这些名称时，编译器通常不会报错，但问题是，这些名称可能与编译器生成的名称冲突，或者与标准库实现在内部使用的名称冲突。注意，这些保留字都具备一个特征：以下划线开头。因此，我们给出如下建议。

提示：

不要使用以下划线开头的名称。

1.4 类和对象

类是定义数据类型的代码块。类的名称就是数据类型的名称。类类型的数据项称为对象。创建变量，以存储自定义数据类型的对象时，就要使用类类型名称。定义自己的数据类型，就可以根据具体的问题提出解决方案。例如，如果编写一个处理学生信息的程序，就可以定义 Student 类型。Student 类型可以包含学生的所有特征，例如年龄、性别或学校记录——这些都是程序需要的。

第 11~14 章将介绍如何创建自己的类，以及如何在程序中使用对象。不过，在那之前，就会使用某些标准库类型的对象，例如第 5 章使用的向量和第 7 章使用的字符串。在技术上讲，甚至 std::cout 和 std::cin 流也是对象。但是，不必担心，使用对象是很简单的，比创建自己的类要简单得多。正因为按照现实生活中的实体来设计对象的行为，所以它们使用起来很直观(不过，也有一些对象是对更加抽象的概念进行的建模，例如输入输出流，或者对低级 C++结构进行的建模，例如数组和字符序列)。

1.5 模板

有时程序需要几个类似的类或函数，其代码中只有所处理的数据类型有区别。编译器可以使用模板给特定的自定义类型自动生成类或函数的代码。编译器使用类模板会生成一个或多个类系列，使用函数模板会生成函数。每个模板都有名称，希望编译器创建模板的实例时，就会使用该名称。标准库大量使用了模板。

第 9 章将介绍如何定义函数模板，第 16 章将介绍如何定义类模板。但是，在之前的章节中，就将使用一些具体的标准库模板，例如第 5 章中会实例化容器类模板，或者使用特定的基础实用函数模板，如 std::min()和 max()。

1.6 代码的表示样式和编程风格

代码排列的方式对代码的可读性有非常重要的影响。这有两种基本的方式。首先，可以使用制表符和/或空格缩进程序语句，显示出这些语句的逻辑；再以一致的方式使用定义程序块的匹配花括号，使程序块(又称代码块、语句块，或简称块)之间的关系更清晰。其次，可以把一个语句放在两行或多行上，提高程序的可读性。

代码有许多不同的表示样式。表 1-1 显示了三种常用的代码表示样式。

表 1-1 三种常用的代码表示样式

样 式 1

```
namespace mine
{
 bool has_factor(int x, int y)
 {
   int factor{ hcf(x, y) };
   if (factor > 1)
   {
     return true;
   }
   else
   {
     return false;
   }
 }
}
```

样 式 2

```
namespace mine {
  bool has_factor(int x, int y)
  {
    int factor{ hcf(x,y) };
    if (factor>1) {
      return true;
    } else {
      return false;
    }
  }
}
```

样 式 3

```
namespace mine {
bool has_factor(int x, int y) {
  int factor{ hcf(x, y) };
  if (factor > 1)
    return true;
  else
    return false;
}
}
```

本书的示例使用样式 1。随着编程经验增加，读者将会根据自己的个人喜好或者公司的要求，形成自己的代码表示样式。建议在某个时候，选择一种适合自己的样式，然后在代码中一致地使用这种样式。一致的代码表示样式不只看上去美观，也会使代码更容易阅读。

对排列匹配花括号和缩进语句所形成的约定，只是编程风格的几个方面之一。其他重要的方面还包括命名变量、类型和函数的约定，以及使用(结构化)注释的约定。良好的编程风格是一个很主观的问题，不过客观来说，有一些指导原则和约定要优于其他。一般来说，采用一致风格的代码更容易阅读和理解，有助于避免引入错误。本书在引导读者形成自己的编程风格时，将不时给出建议。

提示：

关于良好的编程风格，我们能给出的最好的提示之一毫无疑问是：为所有变量、函数和类型选择清晰的描述性名称。

1.7 创建可执行文件

从 C++源代码中创建可执行的模块需要三个步骤。第一步是预处理器处理所有的预处理指令。一般来说，它的关键任务之一是将所有#include 头文件的完整内容复制到.cpp 文件中。第 10 章将讨论其他预处理指令。第二步是编译器把每个.cpp 文件转换为对象文件，其中包含了与源文件内容对应的机器码。第三步是链接程序把程序的对象文件合并到包含完整可执行程序的文件中。

图 1-2 表明，3 个源文件经过编译后，生成 3 个对应的对象文件(图 1-2 中没有明确显示预处理阶段)。用于标识对象文件的文件扩展名在不同的机器环境中是不同的，这里没有显示。组成程序的源文件可以在不同的编译器运行期间单独编译，但大多数编译器都允许在一次运行期间编译它们。无论采用哪种方式，编译器都把每个源文件看作一个独立的实体，为每个.cpp 文件生成一个对象文件。然后在链接步骤中，把程序的对象文件和必要的库函数组合到一个可执行文件中。

在本书的前半部分，程序只包含一个源文件。第 10 章将介绍如何创建一个大程序，在其中包含多个头文件和源文件。

注意：

将源代码转换为可执行文件所需要执行的具体步骤，在各个编译器中是不同的。虽然我们的示例大部分都很小，

可使用一系列的命令行指令来编译和链接，但是使用集成开发环境(Integrated Development Environment，IDE)可能更方便。现代 IDE 提供对用户很友好的图形用户界面，供用户编辑、编译、链接、运行和调试程序。Apress 网站(www.apress.com/book/download.html)提供了最流行的编译器和 IDE 的参考和基本使用说明，以及所有示例代码和练习题的答案。

实际上，编译是一个迭代的过程，因为在源代码中总是会有输入错误或其他错误。更正了每个源文件中的这些错误后，就可以进入链接步骤，但在这一步可能会发现更多的错误！即使链接步骤生成了可执行模块，程序仍有可能包含逻辑错误，即程序没有生成希望的结果。为了更正这些错误，必须回过头来修改源代码，再编译。这个过程会继续下去，直到程序按照希望的那样执行为止。如果程序的执行结果不像我们宣称的那样，其他人就有可能找到我们本应发现的许多错误。一般认为，如果程序超过一定的规模，就总是包含错误，尽管这一点并不能被确切无疑地证实。在乘飞机的时候，最好不要想到这一点。

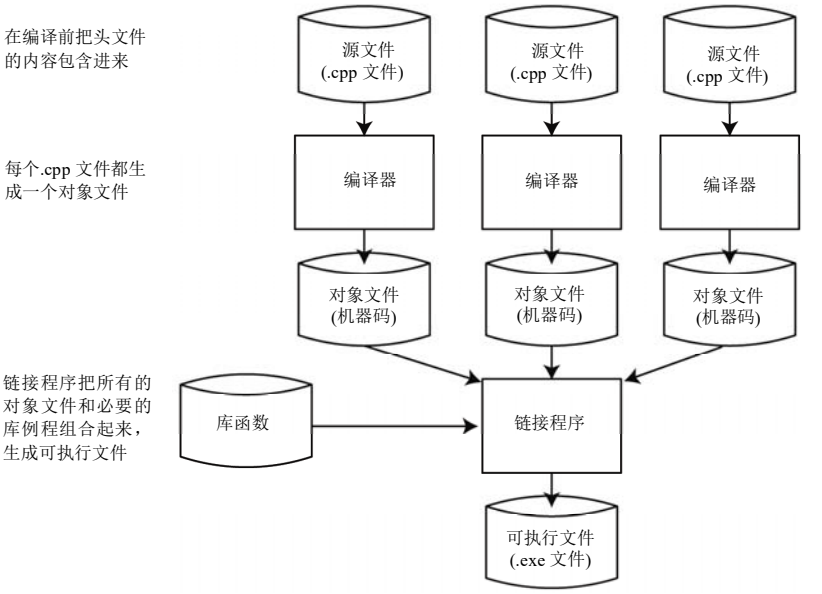

图 1-2　编译和链接过程

1.8　过程化编程和面向对象编程

历史上，过程化编程曾是编写几乎所有程序的方式。要创建问题的过程化编程解决方案，必须考虑程序实现的过程，才能解决问题。一旦需求明确地确定下来，就可以写出完成任务的大致提纲，如下所示：

- 为程序要实现的整个过程进行清晰的说明。
- 将整个过程分为可工作的计算单元，这些计算单元应尽可能是自包含的。它们常常对应于函数。
- 根据正处理的数据的基本类型(如数值数据、单个字符和字符串)来编写函数。

在解决相同问题时，除了开始时对问题进行清晰的说明这一点相同之外，面向对象编程方式在其他地方都完全不同：

- 根据问题的详细说明确定该问题所涉及的对象类型。例如，如果程序处理的是棒球运动员，就应把 BaseballPlayer 标识为程序要处理的数据类型。如果程序是一个会计程序包，就应定义 Account 类型和 Transaction 类型的对象。还要确定程序需要对每种对象类型执行的操作集。这将产生一组与应用程序相关的数据类型，用于编写程序。
- 为问题需要的每种新数据类型生成一个详细的设计方案，包括可以对每种对象类型执行的操作。
- 根据已定义的新数据类型及其允许的操作，编写程序的逻辑。

面向对象解决方案的程序代码完全不同于过程化解决方案，理解起来也比较容易，维护也方便得多。面向对象解决方案所需的设计时间要比过程化解决方案长一些。但是，面向对象程序的编码和测试阶段比较短，问题也比较

少，所以这两种方式的整个开发时间大致相同。

下面简要论述面向对象编程方式。假定要实现一个处理各种盒子的程序。这个程序的一个合理要求是把几个小盒子装到另一个大一些的盒子中。在过程化程序中，需要在一组变量中存储每个盒子的长度、宽度和高度。包含几个盒子的新盒子的尺寸必须根据每个被包含盒子的尺寸，按照为打包一组盒子而定义的规则进行计算。

面向对象解决方案首先需要定义 Box 数据类型，这样就可以创建变量，引用 Box 类型的对象，并创建 Box 对象。然后定义一个操作，把两个 Box 对象加在一起，生成包含前两个 Box 对象的第三个 Box 对象。使用这个操作，就可以编写如下语句：

```
bigBox = box1 + box2 + box3;
```

在这个语句中，+操作的含义远远超出简单的相加。+运算符应用于数值，会像以前那样工作，但应用于 Box 对象时，就有了特殊的含义。这个语句中每个变量的类型都是 Box，上述代码会创建一个 Box 对象，其尺寸足够包含 box1、box2 和 box3。

编写这样的语句要比分别处理所有的尺寸容易得多，计算过程越复杂，面向对象编程方式的优点就越明显。但这只是一个很粗略的说明，对象的功能要比这里所描述的强大得多。介绍这个例子的目的是让读者对使用面向对象方法来解决问题有大致的了解。面向对象编程基本上是根据问题涉及的实体来解决问题，而不是根据计算机喜欢使用的实体(即数字和字符)来解决问题。

1.9 表示数字

在 C++程序中，数字的表示有许多方式，所以必须理解表示数字的各种可能性。如果很熟悉二进制数、十六进制数和浮点数的表示，就可以跳过本节。

1.9.1 二进制数

首先考虑一下常见的十进制数(如 324 或 911)表示什么。显然，324 表示三百二十四，911 表示九百一十一。这是“三百”加上“二十”再加上“四”的简写形式，或是“九百”加上“一十”再加上“一”的简写形式。更明确地说，这两个数表示：

- 324 是 $3\times10^2+2\times10^1+4\times10^0$，也就是 $3\times10\times10+2\times10+4$
- 911 是 $9\times10^2+1\times10^1+1\times10^0$，也就是 $9\times10\times10+1\times10+1$

这称为十进制表示法，因为它建立在 10 的幂的基础之上。也可以说，这里的数字以 10 为基数来表示，因为每个数位都是 10 的幂。以这种方式表示数字非常方便，因为人有 10 根手指和 10 根脚趾可用来计数。但是，这对计算机就不太方便了，因为计算机主要以开关为基础，即开和关，加起来只有 2，而不是 10。这就是计算机用基数 2 而不是用基数 10 来表示数字的主要原因。用基数 2 来表示数字称为二进制计数系统。用基数 10 表示数字，数字可以是 0～9。一般情况下，以任意 n 为基数来表示的数，每个数位的数字是从 0 到 $n-1$。因此，二进制数字只能是 0 或 1，二进制数 1101 就可以分解为：

- $1\times2^3+1\times2^2+0\times2^1+1\times2^0$，也就是 $1\times2\times2\times2+1\times2\times2+0\times2+1$

计算得 13(十进制系统)。在表 1-2 中，列出了用 8 个二进制数字表示的对应的十进制值(二进制数字常常称为位)。

表 1-2 与 8 位二进制值对应的十进制值

二 进 制	十 进 制	二 进 制	十 进 制
0000 0000	0	1000 0000	128
0000 0001	1	1000 0001	129
0000 0010	2	1000 0010	130
…	…	…	…
0001 0000	16	1001 0000	144
0001 0001	17	1001 0001	145
…	…	…	…

(续表)

二 进 制	十 进 制	二 进 制	十 进 制
0111 1100	124	1111 1100	252
0111 1101	125	1111 1101	253
0111 1110	126	1111 1110	254
0111 1111	127	1111 1111	255

使用前 7 位可以表示 0～127 的正数，一共 128 个不同的数，使用全部 8 位可以表示 256(即 2^8)个数。一般情况下，如果有 n 位，就可以表示 2^n 个整数，正值为 0~2^n - 1。

在计算机中，将二进制数相加是非常容易的，因为对应数字加起来的进位只能是 0 或 1，所以处理过程会非常简单。图 1-3 中的例子演示了两个 8 位二进制数相加的过程。

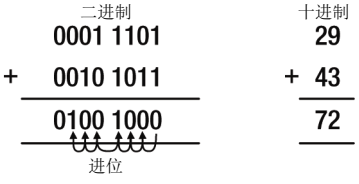

图 1-3 二进制数的相加

相加操作从最右边开始，将操作数中对应的位相加。图 1-3 指出，前 6 位都向左边的下一位进 1，这是因为每个数字只能是 0 或 1。计算 1+1 时，结果不能存储在当前的位中，而需要在左边的下一位中加 1。

1.9.2 十六进制数

在处理很大的二进制数时，就会有一个小问题。例如：

- 1111 0101 1011 1001 1110 0001

在实际应用中，二进制表示法显得比较烦琐，如果把这个二进制数表示为十进制数，结果为 16,103,905，只需要 8 个十进制数位而已。显然，我们需要一种更高效的方式来表示这个数，但十进制并不总是合适的。有时需要能够指定从右开始算起的第 10 位和第 24 位的数字为 1。用十进制整数来完成这个任务是非常麻烦的，而且很容易出现计算错误。比较简单的解决方案是使用十六进制表示法，即数字以 16 为基数表示。

基数为 16 的算术就方便得多，它与二进制也相得益彰。每个十六进制的数字可以是 0～15 的值(10～15 的数字用 A～F 或 a～f 表示，如表 1-3 所示)，0～15 的数值就分别对应于用 4 个二进制数字表示的值。

表 1-3 将十六进制数表示为十进制数和二进制数

十 六 进 制	十 进 制	二 进 制
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A 或 a	10	1010
B 或 b	11	1011

(续表)

十 六 进 制	十 进 制	二 进 制
C 或 c	12	1100
D 或 d	13	1101
E 或 e	14	1110
F 或 f	15	1111

因为一个十六进制数对应于 4 个二进制数，所以可以把较大的二进制数表示为一个十六进制数，方法是从右开始，把每 4 个二进制数组成一组，再用对应的十六进制数表示每个组。例如，二进制数：

- 1111 0101 1011 1001 1110 0001

如果依次提取每 4 个二进制数，用对应的十六进制数表示每个组，将这个数字用十六进制表示，就得到：

- F 5 B 9 E 1

所得的 6 个十六进制数分别对应于 6 组 4 个二进制数。为了证明这适用于所有的情况，下面用十进制表示法把这个数字直接从十六进制转换为十进制。这个十六进制数的计算如下：

- $15\times16^5+5\times16^4+11\times16^3+9\times16^2+14\times16^1+1\times16^0$

最后相加的结果与把二进制数转换为十进制数的结果相同：16,103,905。在 C++中，十六进制数要加上前缀 0x 或 0X，所以在代码中，这个值写作 0xF5B9E1。显然，这意味着 99 与 0x99 不相等。

十六进制数的另一个非常方便的特点是，现代计算机把整数存储在偶数字节的字中，一般是 2、4、8 或 16 字节，一个字节是 8 位，正好是两个十六进制数，所以内存中的任意二进制整数总是精确对应于若干个十六进制数。

1.9.3 负的二进制数

二进制算术需要理解的另一个方面是负数。前面一直假定所有的数字都是正的。乐观地看是这样，所以我们目前已对二进制数有了一半的认识。但在实际中还会遇到负数，悲观地看，我们对二进制数的认识仅仅达到一半。在现代计算机中，是如何表示负数的？稍后将看到，这个问题看起来很简单，但是并不容易直观地给出一个答案。

既可以是正数，又可以是负数的整数称为带符号整数。自然，我们只能使用二进制数位来表示数字。计算机使用的任何语言最终都只会由位和字节组成。因为计算机的内存由 8 位字节组成，所以二进制数字要存储在多个 8 位中(通常是 2 的幂)，即有些数字是 8 位，有些数字是 16 位，有些数字是 32 位，等等。

因此，带符号整数的一种直观的表示是使用固定数量的二进制数位，并指定其中一位作为符号位。在实际应用中，总是会选择最左边的位作为符号位。假设我们把左右带符号整数的大小固定为 8 位。这样一来，数字 6 可表示为 00000110，而－6 则可表示为 10000110。将+6 改为－6，只需要将符号位从 0 改为 1。这称为符号幅值表示法：每个数字都由一个符号位和给定的位数组成，其中符号位为 0 表示正数，符号位为 1 表示负数，给定的位数指定数字的幅值或绝对值，换言之，就是无符号的数字。

虽然符号幅值表示法对人来说是很容易使用的，但是有一个缺点：计算机不容易处理这种表示方法。更具体来说，这种表示方法需要大量的系统开销，需要复杂的电路来执行算术运算。例如，当两个带符号整数相加时，不想让计算机检查两个数字是否为负。我们希望使用简单的、快速的“加”电路来生成相应的结果，而不考虑操作数的符号。

当把 12 和－8 的符号幅值表示简单相加，会发生什么？可以预想到，这种相加操作不会得到正确的结果，但我们还是进行运算。

将 12 转换为二进制： 0000 1100

将－8 转换为二进制(读者可能认为是)： 1000 1000

如果把它们加起来，结果是： 1001 0100

答案是–20，这可不是我们希望的结果+4，它的二进制应是 0000 0100。此时读者会认为，“不能把符号简单地作为另一个位”。但是，为了加快二进制计算，恰恰想要这么做。

因此，几乎所有现代计算机都采用一种不同的方法：使用二进制负数的 2 的补码表示。使用这种表示时，通过一个可以在头脑中完成的简单过程就可以从任何正的二进制数获得其负数形式。现在，我们需要读者信任我们，因

为我们不会解释为什么这种表示法有效。好比一位真正的魔术师，我们不会解释我们的魔术。下面看看如何从正数中构建负数的 2 的补码形式，读者也可以自己证明这是有效的。现在回到前面的例子，给–8 构建 2 的补码形式。

(1) 首先把+8 转换为二进制：

0000 1000

(2) 现在反转每个二进制数字，即把 0 变成 1、把 1 变成 0：

1111 0111

这称为 1 的补码形式。

(3) 如果给这个数加上 1，就得到了 – 8 的 2 的补码形式：

1111 1000

注意，这种方法是双向的。要将负数的 2 的补码形式转换为对应的二进制正数，只需要再次反转所有的位，然后加 1。例如，反转 1111 1000 得到 0000 0111，加 1 后得到 0000 1000，即+8。

当然，如果 2 的补码不能辅助二进制运算，就只能算是一种头脑游戏。我们来看看计算机如何使用 1111 1000：

将+12 转换为二进制形式：　　0000 1100

– 8 的 2 的补码形式是：　　1111 1000

把这两个数加在一起，得到：　　0000 0100

答案就是 4。这是正确的。左边所有的 1 都向前进位，这样该位的数字就是 0。最左边的数字应进位到第 9 位，即第 9 位应是 1，但这里不必担心这个第 9 位，因为在前面计算 – 8 时从前面借了一位，在此正好抵消。实际上，这里做了一个假定，符号位 1 或 0 永远放在最左边。读者可以自己试验几个例子，就会发现这种方法总是有效的。最妙的是，使用负数的 2 的补码形式使计算机上的算术计算——不只是加法——非常简单快速。这是计算机如此擅长计算数字的原因之一。

1.9.4 八进制数

八进制数是用以 8 为基数来表示的数。八进制的数字是 0 到 7。目前八进制数很少使用。计算机内存用 36 位字来衡量时，八进制数很有用，因为可以把 36 位的二进制值指定为 12 个八进制数字。这是很久以前的情形了，为什么还要介绍八进制？因为它可能会引起潜在的混淆。在 C++中仍可以编写八进制的常量。八进制值有一个前导 0，所以 76 是十进制值，076 是八进制值，它对应十进制的 62。下面是一条黄金规则。

警告：

不要在十进制整数的前面加上前导 0，否则会得到另一个值。

1.9.5 Big-Endian 和 Little-Endian 系统

整数在内存的一系列连续字节中存储为二进制值，通常存储为 2、4、8 或 16 个字节。字节采用什么顺序是非常重要的。

把十进制数 262657 存储为 4 字节二进制值。选择这个值是因为它的二进制值是：

00000000 00000100 00000010 00000001

每个字节的位模式都很容易与其他字节区分开来。对于使用 Intel 处理器的 PC(个人电脑)，该数字就存储为如下形式。

字节地址：	00	01	02	03
数据位：	00000001	00000010	00000100	00000000

可以看出，值中最高位的 8 位是都为 0 的那些位，它们都存储在地址最高的字节中，换言之，就是最右边的字节。最低位的 8 位存储在地址最低的字节中，即最左边的字节。这种安排形式称为 Little-Endian。读者可能感到奇怪，为什么计算机要颠倒这些字节的顺序？这么设计的动机依然在于实现更高效的计算和更简单的硬件。具体细节不重要，重要的是应该知道，如今的大部分现代计算机都使用这种与直觉相反的编码方法。

但是，只是大部分计算机采用这种编码方法，也有一部分例外情况。对于使用 Motorola 处理器的机器，该数字在内存中的存储更加符合逻辑。

字节地址：	00	01	02	03

数据位： 00000000 00000100 00000010 00000001

字节现在的顺序是反的，最重要的 8 位存储在最左边的字节中，即地址最低的字节。这种安排形式称为 Big-Endian。一些处理器，例如 Power-PC 处理器以及所有近来生产的 ARM 处理器，采用的都是 Big-Endian 系统，这表示数据的字节顺序可以在 Big-Endian 和 Little-Endian 之间切换。

注意：

无论字节顺序是 Big-Endian 还是 Little-Endian，在每个字节中，最高位都放在左边，最低位都放在右边。

读者也许认为 Big-Endian 还是 Little-Endian 看起来非常有趣，但跟我有什么关系？实际上在大多数情况下，即使不知道执行代码的计算机是采用 Big-Endian 还是 Little-Endian 系统，都可以编写出有效的 C++程序。但是，在处理来自另一台机器的二进制数据时，这就很重要了。二进制数据会写入文件或通过网络传送为一系列字节，此时必须解释它们。如果数据源所在的机器使用的字节顺序与运行代码的机器不同，就必须反转每个二进制数据的字节顺序，否则就会出错。

1.9.6 浮点数

所有整数都是数字，但并不是所有数字都是整数：3.1415 不是整数，0.00001 也不是。许多程序都需要在某个时候处理小数。显然，还需要有一种方式在计算机上表示小数，并且需要能够高效地对小数执行运算。几乎所有计算机都支持一种机制来处理小数，这种机制就是浮点数。

不过，浮点数并不只是代表小数，还可以处理极大的数字。例如，可以使用浮点数来表示宇宙中的质子数，它需要 79 位十进制数字。诚然，这个例子有些极端，但是显然在许多情况下，要处理的数都不仅仅是 32 位二进制整数所能表达的 10 位十进制数字，甚至 64 位二进制整数所能表达的 19 位十进制数字。同样，还有许多非常小的数，例如，汽车销售人员认可你对 2001 款本田汽车的报价所需的分钟数。浮点数能够相当有效地表达这两类数字。

我们首先解释使用十进制浮点数的基本原则。当然，计算机使用的是二进制表示，但是对我们来说，使用十进制时，理解概念要容易多了。所谓的“规格化”数包含两个部分：尾数(或者叫小数)以及指数。这两个部分都可以是正数或负数。数字的幅值是尾数与 10 的指数次方相乘的结果。类似于计算机使用的二进制浮点数表示，我们还将调整尾数和指数的十进制数字的位数。

演示这种表示法要比描述它更容易，所以下面看一些例子。365 写成浮点数的形式为：

```
3.650000E02
```

这里的尾数有 7 位数字，指数有两位。E 表示“指数”，其后是 10 的幂，将尾数部分 3.650000 乘以 10 的幂，就得到需要的值。即，要得到常规的十进制表示法，只需要求出下面算式的积：

$$3.650000 \times 10^{2}$$

这显然是 365。

下面看一个小的数字：

```
-3.650000E-03
```

它计算为 -3.65×10^{-3}，即 -3.65×10^{-3}。将它们称为浮点数，原因非常明显：小数点是浮动的，其位置取决于指数值。

现在假定有一个较大的数字 2, 134, 311, 179。使用相同的数字位数，它表示为：

```
2.134311E09
```

它们不完全相同，因为丢掉了 3 个低位数字，初始值就近似表示为 2, 134, 311, 000。这是处理如此大范围的数字所付出的代价：并不是所有数字都可以用全精度表示；浮点数一般只是精确数字的近似表示。

除了固定精度限制带来的精确度问题之外，还有一个方面要注意。对不同量级的数字执行相加或相减操作时要特别小心。这个问题可以用一个简单的例子来说明。把 1.23E－4 和 3.65E+6 加起来。精确的结果当然是 3,650,000 + 0.000123，即 3,650,000.000123。但是当转换为 7 位精度的浮点数时，就得到了下面的结果：

```
3.650000E+06 + 1.230000E-04 = 3.650000E+06
```

把后面这个较小的数字加到前面那个较大的数字，没有产生效果，和没有执行加法运算没有区别。产生这个问题的原因在于结果只有 7 位小数精度。较大数的所有位数都不会受到较小数的影响，因为较大数的所有有效位数都离较小数的有效位数太远了。

有趣的是，当两个数几乎相等时，也必须特别小心。如果计算这两个数之差，大部分数字的效果是彼此抵消，结果可能只有一两位小数精度。这被称为“灾难性抵消”，这种情况下，很容易计算两个完全垃圾的值。

浮点数可以执行没有它们就不可能执行的计算，但要确保结果有效，就必须记住它们的限制。这意味着考虑要处理的值域及相对值。分析并最大化数学计算和算法的精度(也叫数值稳定性)的领域称为数值分析。但这是一个高级主题，不在本书讨论范围内。只需要知道这一点就够了：浮点数的精度是有限的，执行的数学运算的顺序和性质对结果的精确度有很大的影响。

当然，计算机不处理十进制数，它处理的是二进制浮点数表示，是位和字节。具体来说，如今几乎所有计算机都使用 IEEE 754 标准规定的编码和计算规则。从左至右，每个浮点数包含一个符号位，后跟固定数量的指数位，最后是编码了尾数的另外一系列位。最常用的浮点数表示是所谓的单精度(1 个符号位，8 个指数位，23 个尾数位，总共有 32 位)和双精度(1 + 11 + 52 = 64 位)浮点数。

浮点数可表示极大的数字范围。例如，单精度浮点数已经可以表示 10^{-38}～10^{+38} 的数字。当然，这种灵活性是有代价的：精度的位数是有限的。前面已经说明了这一点，这其实也是符合逻辑的；使用 32 位当然无法精确表示 10^{+38} 量级的所有数字的 38 个位。毕竟，32 位二进制整数所能够精确表示的最大带符号整数只是 $2^{31}-1$，大约是 $2 \times 10^{+9}$。浮点数的小数位的精度位数取决于为其位数分配的内存。例如，单精度浮点数能提供约 7 位小数的精度。这里说“大约”，是因为 23 位的二进制小数不会精确地对应于有 7 位小数位数的十进制小数。双精度浮点数一般对应于约 16 位小数的精度。

1.10 表示字符

计算机中的数据没有内在的含义。机器码指令只是数字，当然数值就是数值，字符也是数值。每个字符都获得了一个独特的整数值，称为代码或代码点。值 42 可以是钼的原子序数，也可以是生活、宇宙和其他事务的答案，还可以是星号字符。这取决于如何解释它。在 C++中，单个字符可以放在单引号中，例如'a' 、'?'、'*'，编译器会给它们生成代码值。

1.10.1 ASCII 码

20 世纪 60 年代，人们定义了美国信息交换标准码(ASCII)来表示字符。这是一个 7 位代码，所以共有 128 个不同的代码值。代码值 0~31 表示各种非打印控制符，例如回车符(代码值 15)和换页符(代码值 12)。代码值 65~90 对应大写字母 A~Z，代码值 97~122 对应小写字母 a~z。如果查看字母的代码值对应的二进制值，就会发现大小写字母的代码值仅在第 6 位上有区别：小写字母的第 6 位是 0，大写字母的第 6 位是 1。其他代码值表示数字 0~9、标点符号和其他字符。

7 位的 ASCII 适合于美国人或英国人，但法国人或德国人在文本中需要重音和元音变音，但它们没有包含在 7 位 ASCII 的 128 个字符中。为了克服 7 位代码的局限，人们定义了 ASCII 的扩展版本，它有 8 位代码。代码值 0~127 与 7 位 ASCII 版本表示相同的字符，代码值 128~255 是可变的。8 位 ASCII 的一种变体称为 Latin-1，它提供大多数欧洲语言中的字符，但也有其他变体用于俄语等语言中的字符。

当然，对于韩国人、日本人、中国人或阿拉伯人而言，8 位 ASCII 肯定是不够的。为了帮助理解，中文、日文和韩文(它们有共同的背景)的现代编码覆盖了大约 88,000 个字符，比 8 位代码能够得到的 256 个字符多得多！为了克服扩展 ASCII 的局限，20 世纪 90 年代出现了通用字符集(Universal Character Set，UCS)，UCS 由标准 ISO 10646 定义，其代码有 32 位，提供了数亿个不同的代码值。

1.10.2 UCS 和 Unicode

UCS 定义了字符和整数代码值(称为代码点)之间的映射。代码点与编码是不同的，认识到这一点很重要：代码点是一个整数，而编码则是把给定的代码点表示为一系列字节或字的方式。小于 256 的代码值非常常见，可以只用

1 个字节表示。如果以固定字节存储，就需要使用 4 字节存储只需要 1 字节的代码值，仅是因为有其他代码值需要多个字节，这是非常低效的。编码是表示代码点，从而允许更高效地存储它们的方式。

Unicode 是一个标准，定义了一组字符及其代码点(与 UCS 相同)。Unicode 还为这些代码点定义了几个不同的编码，包括其他机制，例如处理从右向左阅读的语言(如阿拉伯语)，代码点的范围足以包含世界上所有语言的字符集，还包含许多其他的图形化字符，例如数学符号，甚至表情符号。大多数语言的字符串可以表示为 16 位代码的一个序列。

Unicode 中一个可能让人混淆的方面是：它提供了多个字符编码方法。最常用的编码是 UTF-8、UTF-16 和 UTF-32，它们都可以表示 Unicode 集合中的所有字符。它们之间的区别是如何表示给定字符的代码点，给定字符的代码值在这三种表示法中是相同的。下面是这些编码表示字符的方式：

- UTF-8 把字符表示为长度在 1 字节和 4 字节之间变化的序列。ASCII 字符集在 UTF-8 中表示为单字节代码，其代码值与 ASCII 相同。大多数网页都使用 UTF-8 编码文本。
- UTF-16 把字符表示为一个或两个 16 位值。UTF-16 包括了 UTF-8。因为一个 16 位值包含 0 代码段中的所有代码，所以 UTF-16 覆盖了多语言编程环境中的大多数情形。
- UTF-32 将所有字符表示为 32 位值。

存储 Unicode 字符有 4 个整数类型可用：char、wchar_t、char16_t 和 char32_t。详见第 2 章。

1.11 C++源字符

编写 C++语句要使用基本源字符集，这些是在 C++源文件中可以显式使用的字符集。用于定义名称的字符集是上述字符集合的一个子集。当然，基本源字符集并没有限制代码中使用的字符数据。程序可以用各种方式创建没有包含在该字符集中的字符串，后面将会看到示例。基本源字符集包括下述字符：

- 大小写字母 A～Z 和 a～z
- 数字 0～9
- 空白字符，如空格、水平和垂直制表符、换页符和换行符
- 字符_ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " '

这很简单、直观。一共可以使用 96 个字符，这些字符可以满足大多数要求。大多数情况下，基本源字符集足够用了，但偶尔需要使用不包含在基本源字符集中的字符。至少在理论上，可以在名称中包含 Unicode 字符。Unicode 字符可指定为其代码点的十六进制表示\udddd 或\Uddddddddd(其中 d 是一个十六进制数)。注意第一种形式使用小写 u，第二种形式使用大写 U，两者都是可接受的。但是，编译器对在名称中使用 Unicode 字符提供的支持是有限的。字符和字符串数据都可以包含 Unicode 字符。

转义序列

在程序中使用字符常量时，例如单个字符或字符串，某些字符是会出问题的。显然，不能直接把换行符输入为字符常量，因为它们只完成自己该做的工作：转到源代码文件中新的一行(唯一的例外是第 7 章将会介绍的原字符串)。通过转义序列可以把这些控制字符输入为字符常量。转义序列是指定字符的一种间接方式，总是以一个反斜杠\开头。表示控制字符的转义序列如表 1-4 所示。

表 1-4 表示控制字符的转义序列

转 义 序 列	控 制 字 符
\n	换行符
\t	水平制表符
\v	垂直制表符
\b	退格符
\r	回车符
\f	换页符
\a	警告字符

还有其他一些字符在直接表示时会出问题。显然，表示反斜杠字符本身是很困难的，因为它表示转义序列的开头。用作界定符的单引号和双引号，例如常量'A'或字符串"text"也有问题(这取决于具体的上下文，后面将详细说明)。表 1-5 列出了这些转义序列。

表 1-5　用转义序列指定的“问题”字符

转 义 序 列	字　　符
\\	反斜杠
\'	单引号
\"	双引号

由于反斜杠表示转义序列的开始，因此把反斜杠字符作为字符常量的唯一方式是使用两个连续的反斜杠(\\)。

下面的程序示例使用转义序列输出要显示在屏幕上的消息。要查看该程序的执行结果，需要输入、编译、链接和执行下面的程序。

```
// Ex1_02.cpp
// Using escape sequences
#include <iostream>

int main()
{
  std::cout << "\"Least \'said\' \\\n\t\tsoonest \'mended\'.\"" << std::endl;
}
```

在编译、链接和运行这个程序时，会显示如下结果：

```
"Least 'said' \
              soonest 'mended'."
```

所得的输出由下面语句中双引号之间的内容确定：

```
std::cout << "\"Least \'said\' \\\n\t\tsoonest \'mended\'.\"" << std::endl;
```

原则上，上述语句中外部双引号之间的所有内容都会发送给 cout。双引号之间的字符串称为字符串字面量。双引号字符是界定符，表示该字符串字面量的开始和结束；它们不是字符串的一部分。字符串字面量中的转义序列会被编译器转换为它表示的字符，所以该字符会发送给 cout，而不是发送转义序列。字符串字面量中的反斜杠总是表示转义序列的开始，所以发送给 cout 的第一个字符是双引号。

接下来输出的是 Least 后跟一个空格，接着是一个单引号、said 和另一个单引号。然后是一个空格和由\\指定的反斜杠。接着把对应于\n 的换行符写入流，使光标移动到下一行的开头。然后用\t\t 给 cout 发送两个制表符，让光标向右移动两个制表位。之后显示字符串 soonest、一个空格和单引号中的 mended，最后是句号和一个双引号。

注意：

如果不喜欢使用转义序列，第 7 章将介绍一种替代方法：原字符串字面量。

事实上，上面 Ex1_02.cpp 使用的字符转义有些冗余。在字符串字面量中，实际上并不需要转义单引号字符，直接使用它并不会导致混淆。因此，下面的语句也会有相同的效果：

```
std::cout << "\"Least 'said' \\\n\t\tsoonest 'mended'.\"" << std::endl;
```

只有用在\"这种形式的字符字面量中时，才真正需要转义单引号。反过来，在这种情况中，就不需要转义双引号了；编译器会接受\"'和'"'。我们有些超前了：字符字面量应该是下一章的主题。

注意：

严格来说，Ex1_02 中的\t\t 转义序列也不是必需的，原则上也可以在字符串字面量中键入制表位(如"\"Least 'said' \\\n soonest 'mended'.\" ")。不过，仍然建议使用\t\t；使用制表位的问题是，一般很难区分制表位" "和多个空格" "，更不必说恰当地统计制表位的数量了。另外，在保存文件时，一些文本编辑器会把制表位转换为空格。因此，风格指南要求在字符串字面量中使用\t 转义序列是十分常见的。

1.12 本章小结

本章简要介绍了 C++的一些基本概念。后面将详细讨论本章提及的内容。本章的要点如下：

- C++程序包含一个或多个函数，其中一个是 main()函数。执行总是从 main()函数开始。
- 函数的可执行部分由包含在一对花括号中的语句组成。
- 一对花括号定义了一个语句块。
- 语句以分号结束。
- 关键字是 C++中有特殊含义的一组保留字。程序中的实体不能与 C++语言中的任何关键字同名。
- C++程序包含在一个或多个文件中。源文件包含可执行代码，头文件包含可执行代码使用的定义。
- 定义函数的代码通常存储在扩展名为.cpp 的源文件中。
- 源文件使用的定义通常存储在扩展名为.h 的头文件中。
- 预处理器指令指定了要对文件中的代码执行的操作。所有的预处理器指令都在编译文件中的代码之前执行。
- 头文件中的代码通过#include 预处理器指令添加到源文件中。
- 标准库提供了支持和扩展 C++语言的大量功能。
- 要访问标准库函数和定义，可以把标准库头文件包含到源文件中。
- 输入和输出是利用流来执行的，需要使用插入和提取运算符，即<<和>>。std::cin 是对应于键盘的标准输入流，std::cout 是把文本写入屏幕的标准输出流。它们都在 iostream 标准库头文件中定义。
- 面向对象编程方式需要定义专用于某问题的新数据类型。一旦定义好需要的数据类型，就可以根据这些新数据类型来编写程序。
- Unicode 定义的独特整数代码值表示世界上几乎所有语言的字符，以及许多专用的字符集。代码值也称为代码点。Unicode 还定义了代码点如何编码为字节序列。

1.13 练习

下面的练习用于巩固本章学习的知识点。如果有困难，可回过头重新阅读本章的内容。如果仍然无法完成练习，可以从 Apress 网站(www.apress.com/source-code)下载答案，但只有别无他法时才应该查看答案。

1. 创建、编译、链接、执行一个程序，在屏幕上输出文本"Hello World"。
2. 创建并执行一个程序，在一行上输出自己的姓名，在下一行上输出年龄。
3. 下面的程序有几处编译错误。请指出这些错误并更正，使程序能正确编译并运行。

```
include <iostream>

Int main()
{
   std:cout << "Hello World" << std:endl
)
```

第 2 章

基本数据类型

本章将介绍每个程序都需要的、C++内置的基本数据类型。C++的面向对象功能全部建立在这些基本数据类型的基础之上，因为用户创建的所有数据类型最终都是根据计算机操作的基本数值数据定义的。学习完本章后，读者将能编写传统格式(输入-处理-输出)的简单 C++程序。

本章主要内容

- C++中的基本数据类型
- 变量的声明和初始化
- 如何固定变量的值
- 整型字面量及其定义
- 计算的过程
- 如何定义包含浮点数的变量
- 如何创建存储字符的变量
- auto 关键字的作用

2.1 变量、数据和数据类型

变量是用户定义的一个命名的内存段。每个变量都只存储特定类型的数据。每个变量都定义了可以存储的数据类型。每个基本类型都用唯一的类型名称(即关键字)来标识。关键字是 C++中的保留字，不能用于其他目的。

编译器会进行大量的检查，确保在给定的上下文中使用正确的数据类型，它还确保在操作中合并不同的类型(例如将两个值相加)时，它们要么有相同的类型，要么可以把一个值转换为另一个值的类型，使它们相互兼容。编译器检测并报告尝试把不兼容的数据类型组合在一起而产生的错误。

数值分为两大类：整数和浮点数(可以是小数)。在每个大类中，都有几种基本的 C++类型，每种类型都可以存储特定的数值范围。首先介绍整数类型。

2.1.1 定义整型变量

下面的语句定义了一个整型变量：

```
int apple_count;
```

这个语句定义了一个 int 类型的变量 apple_count，该变量包含某个随机的垃圾值。在定义变量时，可以而且应该指定初始值，如下所示：

```
int apple_count {15};                              // Number of apples
```

apple_count 的初始值放在变量名后面的花括号中，所以其值为 15。包含初始值的花括号称为初始化列表。在本书后面，初始化列表常常包含几个值。定义变量时不必初始化它们，但最好进行初始化。确保变量一开始就有已知的值，在代码不像预期的那样工作时，将便于确定出错的位置。