C 语言入门经典

(第 6 版)

[智] 杰曼 • 冈萨雷斯 • 莫里斯(German Gonzalez-Morris) 著
[英] 艾弗 • 霍顿(Ivor Horton)
童　晶　　李天群　　译

清华大学出版社
北　京

北京市版权局著作权合同登记号　图字：01-2021-5385

图书在版编目(CIP)数据

C语言入门经典：第6版 / (智)杰曼 • 冈萨雷斯 • 莫里斯，(英)艾弗 • 霍顿著；童晶，李天群译. —北京：清华大学出版社，2021.9

书名原文：Beginning C: From Beginner to Pro, Sixth Edition

ISBN 978-7-302-59026-2

Ⅰ. ①C…　Ⅱ. ①杰… ②艾… ③童… ④李…　Ⅲ. C 语言—程序设计　Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2021)第 177127 号

责任编辑：王　军
装帧设计：孔祥峰
责任校对：成风进
责任印制：

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 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张**：37.5　　**字　　数**：1033 千字
版　　次：2021 年 10 月第 1 版　　**印　　次**：2021 年 10 月第 1 次印刷
定　　价：139.00 元

产品编号：090584-01

译 者 序

随着人工智能时代的到来，学习编程已经成为国内的一种学习热潮。而 C 语言凭借其自身简洁、灵活和功能强大的特点，近年来一直占据着流行编程语言的前茅。C 语言是许多高级计算机语言的基础，学好 C 语言能更好地学习其他高级语言。C 语言的应用范围非常广泛，具备很强的数据处理能力，不仅仅是在软件开发上，而且各类科研都需要用到 C 语言，适于编写系统软件、三维与二维图形和动画，具体应用如单片机以及嵌入式系统开发。

目前市面上关于 C 语言学习的图书很多，本书以深入浅出的方法介绍 C 语言中抽象的语法和算法，非常适合初学者编程入门学习。同时，本书知识结构清晰，内容详细，也可作为有经验的程序员的枕边书，随时可以查阅解惑。在 IT 领域，我想大多数程序员精英都读过 Ivor Horton 的图书，本书作者 Ivor Horton 是世界著名的计算机图书作家，帮助无数程序员步入编程的殿堂。时间推移、日月更替，作为 C 语言入门的经典图书，《C 语言入门经典》已历经多次版本迭代，译者翻译的是《C 语言入门经典》的第 6 版。

科技的进步使人们的生活变得更加丰富多彩，但是编程的学习却是比较枯燥的，因此也有很多编程初学者“无疾而终”。此处译者想给编程初学者几个小小的建议，希望对读者的编程学习生涯有所帮助。

1. 兴趣是最好的老师，但大部分学习者对编程的学习可能一开始并没有很大的兴趣。那就需要体会编程带来的成就感，例如成功地执行了一个程序，成功地找到了一个 bug，都会让人感觉很有成就，需要享受这种编程带来的成就感。

2. 遇到问题时不要轻言放弃，可以先尝试自己找出问题进行分析。如果不行，就网页搜索看看有没有解决方案，还是不行，可以询问认识的朋友。一般经历这几个过程，绝大部分的问题都可以得到解决。

3. 学会调试，遇到程序出错时，可以利用一些简单的功能让程序输出你想看到的内容，也可以利用编辑器的调试功能进行调试。

4. 在学习的过程中，可以将学习的心得及过程用博客的方式记录下来，一方面可以回溯自己的学习过程，一方面可以帮助后来的初学者参考。

5. “纸上学来终觉浅”，编程的学习最重要的就是要动手去写代码，而不是看代码。希望读者在学习的过程中，对本书中的每一个案例都能动手实现一遍。对于课后练习，也要认真编程完成，方能将理论与实践相结合，真正地掌握编程的知识。

学习没有捷径，唯有脚踏实地、认真钻研，方能成为真正的编程精英，领略编程之美、计算机之美。

在这里，我需要感谢参与本书翻译工作的李天群，感谢清华大学出版社的编辑以及所有参与本书校对的人员。为保证本书的质量，你们帮助我解决了很多问题。

机缘巧合之下，我接到翻译本书的任务，怀着对大师的敬仰之情，基于第 5 版的翻译基础，译者在翻译过程中力求“信、达、雅”，历时半年，终于完成对本书的翻译工作。但由于译者水平有限，在翻译过程中难免有一些疏漏之处，请读者不吝指正。

童　晶

作 者 简 介

German Gonzalez-Morris 是一名 C/C++、Java 和开发不同应用程序容器的软件设计师/工程师，特别专注在 WebLogic 服务器方面的工作。他还从事开发不同的应用程序，包括 JEE/Spring/Python。他的工作领域还包括 OOP、Java/JEE、Python、设计模式、算法、Spring Core/MVC/Security 和微服务。German 曾在消息传递性能、RESTful API 和事务系统方面工作过。

Ivor Horton 是一家从事咨询业的自营职业者，撰写编程方面的教程。他在 IBM 工作多年。Ivor 在 IBM 的工作包括在各种机器上用大多数语言(如汇编语言和高级语言)编程、实时编程以及设计和实现实时闭环工业控制系统。他在培训工程师和其他专家学习编程(Fortran、PL/1、APL 等)方面有着丰富的经验。Ivor 是机械、工艺和电子 CAD 系统、机械 CAM 系统和 DNC/CNC 系统方面的专家。

技术审稿人简介

Michael Thomas 作为独立贡献者、团队负责人、项目经理和工程副总裁在软件开发领域工作了 20 多年。Michael 有超过 10 年的移动设备工作经验。他目前的工作重点是医疗领域，利用移动设备加快患者和医疗保健提供者之间的信息传输。

致　　谢

我要感谢我的家人——我的父母 Germán 和 Felicia Morris 给了我受教育的机会和支持；我的爱人 Patricia Cruces 给了我无限的耐心和爱；我的儿子 Raimundo 和 Gregorio 给了我幸福和灵感。

我很珍惜整个 Apress 团队、Steve Anglin 和 Mark Powers 给予我的机会和支持，并感谢他们的指导和建议。我还要感谢技术评审员 Michael Thomas 提供了重要的反馈、建议和一些错误更正。

感谢我的同事 Ariel Aguayo、Carlos Hasan 和 Daniel Lagos 对书中内容给出看法和建议。

前　言

欢迎使用本书，研读本书，你可以成为一位称职的 C 语言程序员。从许多方面来说，C 语言都是学习程序设计的理想起步语言。C 语言很简洁，因此不必学习大量的语法便能够开始编写真正的应用程序。除了简明易学以外，它还是一门功能非常强大的语言，并被专业人士广泛应用在各种领域。C 语言的强大之处主要体现在，它能够应用于各类层次的开发中；从设备驱动程序和操作系统组件到大规模应用程序，它都能胜任。此外，C 语言还适用于较新的手机应用程序开发。

几乎所有计算机都包含 C 语言编译器，因此，当你学会了 C 语言，就可以在任何环境下进行编程。最后一点，掌握 C 语言可以为理解面向对象的 C++语言奠定良好的基础。

在作者眼中，有抱负的程序员必将面对三重障碍，即掌握遍布程序设计语言中的各类术语、理解如何使用语言元素(而不仅仅只是知道它们的概念)，以及领会如何在实际场景中应用该语言。本书的目的就是将这些障碍降到最低限度。

术语是专业人士及优秀业余爱好者之间的交流必不可少的，因此有必要掌握它们。本书将确保你理解这些术语，并自如地在各种环境下使用它们。这样才能更有效地使用大多数软件产品附带的文档，且能轻松地阅读和学习大部分程序设计语言相关的著作。

理解语言元素的语法和作用固然是学习 C 语言过程中的一个重要部分，但认识语言特性如何工作及应用同等重要。本书不仅采用了代码片段，还在每个章节中使用一些实际应用示例展示语言特性如何应用于特定的问题。这些示例提供了实践的基础，读者可以通过改动代码观察修改后的结果。

理解特定背景下的程序设计不只是应用个别语言元素。为了帮助读者理解它们，本书大部分章节之后都给出了一个较复杂的应用程序，以应用本章之前学到的知识。这些程序可以帮助你获得开发应用程序的能力与信心，了解如何组合以及更大范围地应用语言元素。最重要的是，它们能让你了解设计实际应用程序与管理实际代码会碰到的问题。

不管学习什么程序设计语言，有几件事情都要意识到。首先，虽然要学的东西很多，但是掌握它们之后，你就会有极大的成就感；其次，学习的过程很有趣，你会深深地体会到这点；第三，只有通过动手实践才能学会编程，这也是本书贯彻的思想。最后，在学习的过程中，肯定会时不时犯许多错误和感到沮丧。当觉得自己完全停滞时，你要做的就是坚持。最终你一定会体验到成功的喜悦，并且回顾时，你会觉得它也并没有你当初想象的那么难。

如何使用本书

作者认为动手实践是学习编程最好的方法，很快你就会编写第一个程序了。每一章都会有几个将理论应用于实践的程序，它们也是本书的核心所在。建议读者手工输入并运行书中的示例，因为手工输入可以极大地帮助记忆语言元素。此外，你还应当尝试解决每章末尾的所有练习题。当你第一次将一个程序运行成功，尤其是在解决自己的问题后，你会有很大的成就感并感觉到惊人的进步，那时你一定会觉得一切都值得。

刚开始，学习的进展不会太快。不过随着逐渐深入，你的学习进度会越来越快。每一章都会

涉及许多基础知识，因此在学习新的内容之前，需要花些时间确保理解前面学习过的所有知识。实践各部分的代码，并尝试实现自己的想法，这是学习程序设计语言的一个重要部分。尝试修改书中的程序，看看还能让它们做些什么，那才是有趣之处。不要害怕尝试，如果某些地方不太明白，尝试输入一些变体，看看会出现什么情况。出错并没什么大不了，你会从出错中学到很多知识。一个不错的方法是通读每一章，了解各章的范围，然后回过头来过一遍所有的示例。

你可能会觉得某些章末尾的练习题非常难。如果第一次没有完全搞明白，不必担心。之所以第一次觉得困难是因为它们通常都是将你所学的知识应用到了相对复杂的问题中。如果你实在觉得困难，那么可以略过它们继续学习下一章，然后再回过头来研究这些程序。你甚至可以阅读完整本书再考虑它们。尽管如此，如果你能完成练习，就说明你取得了真正的进步。

本书读者对象

本书的目的是教会读者如何尽可能简单快速地编写有用的程序。在阅读完全书后，读者会完全了解 C 语言编程。这本教程面向的是那些之前编过一些程序，了解背后的概念，并且希望通过学习 C 语言进一步扩展知识的读者。尽管如此，本书并未假设读者拥有先前的编程知识，因此如果你刚刚接触编程，本书依然是你的不错选择。

使用本书的条件

要使用本书，你需要一台安装 C 编译器和库的计算机以执行书中的示例，以及一个程序文本编译器用于创建源代码文件。你使用的编译器应支持目前 C 语言国际标准 C17(ISO/IEC 9899:2011，是 C11 的错误修复版本)。你还需要一个用于创建和修改代码的编辑器，可以采用纯文本编辑器(如记事本或 vi)创建源文件。不过，采用专为编辑 C 语言代码设计的编辑器会更有帮助。

以下是作者推荐的两款 C 语言编译器，均为免费软件。

- GNU C 编译器(GCC)，可从 www.gnu.org 下载，它支持多种不同的操作系统环境。
- 面向 Microsoft Windows 的 Pelles C 编译器，可从 www.smorgasbordet.com/pellesc/下载，它提供了一个非常棒的集成开发环境(IDE)。

本书采用的约定

本书的文本和布局采用了许多不同的样式，以便区分各种不同的信息。大多数样式表达的含义都很明显。程序代码样式如下：

```
int main(void)
{ printf("Beginning C\n");
  return 0;
}
```

如果代码片段是从前面的实例修改而来，那么修改过的代码行就用粗体显示，如下所示。

```
int main(void)
{
  printf("Beginning C by Ivor Horton\n");
   return 0;
}
```

当代码出现在文本中时，它的样式会有所不同，如 double。

程序代码中还使用了各种“括号”。本书中称()为圆括号，{}为花括号，[]为方括号。

目　录

第 1 章

C 语言编程

C 语言是一种功能强大、简洁的计算机语言，通过它可以编写程序，指挥计算机完成指定的任务。我们可以利用 C 语言创建程序(即一组指令)，并让计算机依指令行事。

用 C 语言编程并不难，本书将用浅显易懂的方法介绍 C 语言的基础知识。读完本章，读者就可以编写第一个 C 语言程序了。其实，C 语言很简单。

本章的主要内容：

- C 语言标准
- 标准库的概念
- 如何创建 C 程序
- 如何组织 C 程序
- 如何编写在屏幕上显示文本的程序

1.1 C 语言

C 语言是相当灵活的，用于执行计算机程序能完成的几乎所有任务，包括会计应用程序、字处理程序、游戏、操作系统等。它不仅是更高级语言(如 C++)的基础，目前还以 Objective C 的形式开发手机应用程序。Objective C 是标准的 C 加上一小部分面向对象编程功能，并增加很多的新设备/微控制器，如树莓派和 Arduino。C 语言很容易学习，因为它很简洁。因此，如果你立志成为一名程序员，最好从 C 语言开始学起，能快速而方便地获得编写实际应用程序的足够知识。

C 语言由一个国际标准定义，目前其最新版本由 C17(ISO/IEC 9899:2018)定义，它是 C11 的错误修复版本，而不是新特性(例如，它不支持 ATOMIC_VAR_INIT)。现行标准通常被称为 C17 或 C18——本版本的非正式名称。这是因为它在 2017 年完成，但在 2018 年出版。众所周知，GCC 将 C17 作为参考来对标新版本。然而，上述内容并未在标准中声明。我在本书中描述的语言符合 C17，或者可以认为 C11 有几个已解决的问题。需要注意的是，C17 定义的一些语言元素是可选的。这表示，遵循 C17 标准的 C 编译器可能没有实现该标准中的所有功能(编译器只是一个程序，它可以把用我们能理解的术语所编写的程序转换为计算机能理解的术语)。本书会标识出 C11 中的可选语言特性，这样读者就知道，自己的编译器可能不支持它。在这本书中我们将 C11/C17 作为同义词使用。

C17 编译器还有可能没有实现 C17 标准强制的所有语言特性。实现新语言功能是需要时间的，

所以编译器开发人员常常采用逐步接近的方式实现它们。这也是程序可能不工作的另一个原因。尽管如此，根据我的经验，C 程序不能工作的最常见原因，至少有 99.9%的可能性是出现了错误。

1.2 标准库

C 的标准库也在 C17 标准中指定。标准库定义了编写 C 程序时常常需要的常量、符号和函数。它还提供了基本 C 语言的一些可选扩展。取决于机器的特性，例如计算机的输入输出，由标准库以不依赖机器的形式实现。这意味着，在个人计算机(PC)中用 C 代码把数据写入磁盘文件的方式，与在其他计算机上相同，尽管底层的硬件处理不同。库提供的标准功能包括大多数程序员都可能需要的功能，例如处理文本字符串或数学计算，这样就免除了自己实现这些功能所需的大量精力。

标准库在一系列标准文件——头文件中指定。头文件的扩展名总是.h。为使一组标准功能可用于 C 程序文件，只需要将对应的标准头文件包含进来，其方式在本章后面介绍。我们编写的每个程序都会用到标准库。附录 E 汇总了构成标准库的头文件。

开始，有一个为 ANSI C 实现了许多特性的 C POSIX 库。其中一个库是 pthreads，它现在已经过时并且在标准库中实现。其他 POSIX 库(ISO/IEC 9945(POSIX))也在 C2x 未来版本的规划中。

1.3 学习 C 语言

如果你对编程非常陌生，则不需要学习 C 的某些方面，至少在刚开始时不需要学习。这些功能比较特殊，或者不大常用。本书把它们放在第 14 章，这样读者可以在熟悉其他内容后，再学习它们。

尽管所有示例代码都可以从网站(www.apress/com/9781484259757)下载，我还是建议读者自己输入本书中的所有示例，即使它们非常简单。自己亲自输入，以后就不容易忘记。不要害怕用代码进行实验。犯错对编程而言非常有教育性。早期犯的错误越多，学到的东西就越多。

1.4 创建 C 程序

C 程序的创建过程有 4 个基本步骤。

- 编辑
- 编译
- 链接
- 执行

这些过程很容易完成。首先介绍每个过程，以及它们对创建 C 程序的作用。

1.4.1 编辑

编辑过程就是创建和修改 C 程序的源代码——我们编写的程序指令称为源代码。有些 C 编译器带一个编辑器，可帮助管理程序。通常，编辑器是提供了编写、管理、开发与测试程序的环境，有时也称为集成开发环境(Integrated Development Environment，IDE)。

也可以使用一般的文本编辑器创建源文件，但它们必须将代码保存为纯文本，而没有嵌入附加的格式化数据。不要使用字处理器(如微软的 Word)，字处理器不适合编写程序代码，因为它们在保存文本时，会附加一些格式化信息。一般来说，如果编译器系统带有编辑器，就会提供很多

更便于编写及组织程序的功能。它们通常会自动编排程序文本的格式，并将重要的语言元素以高亮颜色显示，这样不仅让程序容易阅读，还容易找到单词输入错误。

在 Linux 上，最常用的文本编辑器是 Vim 编辑器，也可以使用 GNU Emacs 编辑器。对于 Microsoft Windows，可以使用许多免费(freeware)或共享(shareware)的程序设计编辑器。这些软件提供了许多功能，例如，高亮显示特殊的语法及代码自动缩进等功能，帮助确保代码是正确的。Emacs 编辑器也有 Microsoft Windows 版本。UNIX 环境的 vi 和 Vim 编辑器也可用于 Windows，甚至可以使用 Notepad++(http://notepad-plus-plus.org/)。

当然，也可以购买支持 C 语言的专业编程开发环境，例如 JetBrains 或 Microsoft(有免费的社区版)的相关产品，它们能大大提高代码编辑能力。不过，在付款之前，最好检查它们支持的 C 级别是否符合当前的 C 语言标准 C17。因为现在很多编辑器产品主要面向 C++开发人员，C 语言只是一个次要目标。

1.4.2 编译

编译器可将源代码转换成机器语言，在编译过程中，会找出并报告错误。这个阶段的输入是在编辑期间生成的文件，常称为源文件。

编译器能找出程序中很多无效或无法识别的错误，以及结构错误，例如程序的某部分永远不会执行。编译器的输出结果称为对象代码(object code)，存放它们的文件称为对象文件(object file)，这些文件的扩展名在 Microsoft Windows 环境中通常是.obj，在 Linux/UNIX 环境中通常是.o。编译器可以在转换过程中找出几种不同类型的错误，它们大都会阻止对象文件的创建。

如果编译成功，就会生成一个文件，它与源文件同名，但扩展名是.o 或者.obj。

如果在 UNIX 系统下工作，在命令行上编译 C 程序的标准命令是 cc(若编译器是 GNU's Not UNIX(GNU)，则命令为.gcc)。下面是一个示例：

```
cc -c myprog.c
```

其中，myprog.c 是要编译的程序，如果省略了-c 这个参数，程序还会自动链接。成功编译的结果是生成一个对象文件。

大多数 C 编译器都有标准的编译选项，在命令行(如 cc myprog.c)或集成开发环境下的菜单选项(Compile 菜单选项)里都可找到。在 IDE 中编译常常比使用命令行容易得多。

编译过程包括两个阶段。第一个阶段称为预处理阶段，在此期间会修改或添加代码；第二个阶段是生成对象代码的实际编译过程(第二阶段在下面进行汇编。GCC 和其他编译器可以选择这些步骤，但大多数时候是不必要的)。源文件可以包含预处理宏，它们用于添加或修改 C 程序语句。如果现在不理解它们，不必担心，本书后面将进行详细论述。

1.4.3 链接

链接器(linker)将源代码文件中由编译器产生的各种对象模块组合起来，再从 C 语言提供的程序库中添加必要的代码模块，将它们组合成一个可执行的文件。链接器也可以检测和报告错误，例如，遗漏了程序的某个部分，或者引用了一个根本不存在的库组件。

实际上，如果程序太大，可将其拆成几个源代码文件，再用链接器连接起来。因为很难一次编写一个很大的程序，也不可能只使用一个文件。如果将它拆成多个小源文件，每个源文件提供程序的一部分功能，程序的开发就容易多了。这些源文件可以分别编译，更容易避免简单输入错误的发生。再者，整个程序可以一点一点地开发，组成程序的源文件通常会用同一个项目名称集

成，这个项目名称用于引用整个程序。

程序库提供的例程可以执行非 C 语言的操作，从而支持和扩展了 C 语言。例如，库中包含的例程支持输入、输出、计算平方根、比较两个字符串，或读取日期和时间信息等操作。

链接阶段出现错误，意味着必须重新编辑源代码；反过来，如果链接成功，就会生成一个可执行文件，但这并不一定表示程序能正常工作。在 Microsoft Windows 环境下，这个可执行文件的扩展名为.exe；在 UNIX 环境下，没有扩展名，但它是一个可执行的文件类型。多数 IDE 也有 Build 选项，它可一次完成程序的编译和链接。

1.4.4 执行

执行阶段就是当成功完成了前述 3 个过程后，运行程序。但是，这个阶段可能会出现各种错误，包括输出错误及什么也不做，甚至使计算机崩溃。不管出现哪种情况，都必须返回编辑阶段，检查并修改源代码。

在这个阶段，计算机最终会精确地执行指令。在 UNIX 和 Linux 下，只要键入编译和链接后的文件名，即可执行程序。在大多数 IDE 中，都有一个相应的菜单命令来运行或执行已编译的程序。这个 Run 命令或 Execute 命令可能有自己的菜单，也可能位于 Compile 菜单项下。在 Windows 环境中，运行程序的.exe 文件即可，这与运行其他可执行程序一样。

在任何环境及任何语言中，开发程序的编辑、编译、链接与执行这 4 个步骤都是一样的。图 1-1 总结了创建 C 程序的各个过程。

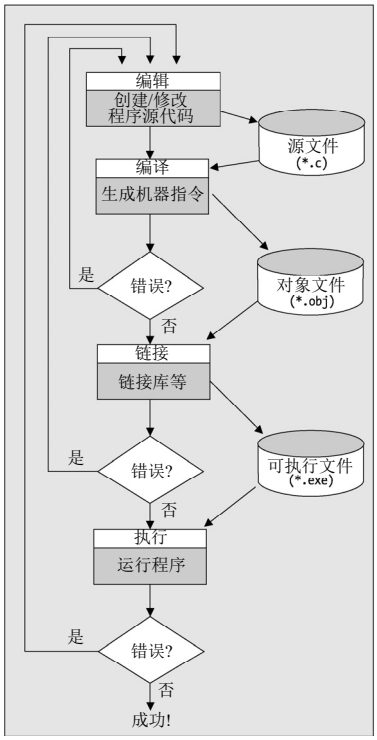

图 1-1　创建与执行一个程序

1.5 创建第一个程序

本节先浏览创建 C 语言程序的流程，从输入代码到执行程序的所有 4 个步骤。在这个阶段，若不了解所键入的代码信息，别担心，笔者会解释每一个步骤。

试试看：C 程序示例

打开编辑器，输入下面的程序，请注意标点符号不要输错，第 4 行及最后一行的括号是花括号{}，而不是方括号[]或者圆括号()——这很重要。另外一定要键入斜杠(/)，以后也会用到反斜杠(\)。最后别忘了行末的分号(;)。

```
/*  Program 1.1 Your Very First C Program - Displaying Hello World */
#include  <stdio.h>

int main(void)
{
   printf("Hello  world! ");
   return 0;
}
```

输入上面的源代码后，将程序保存为 hello.c。可以用任意名字替代 hello，但扩展名必须是.c。这个扩展名在编写 C 程序时是一个通用约定，它表示文件的内容是 C 语言源代码。大多数 C 编译器都要求源文件的扩展名是.c，否则编译器会拒绝处理它。

下面编译程序(如本章“编译”一节所述)，链接所有必要的内容，创建一个可执行程序(如前面“链接”一节所述)。编译和链接一般在一个操作中完成，通常称为“构建操作”。源代码编译成功后，链接器就添加程序需要的标准库代码，为程序创建一个可执行文件。

最后，执行程序。这有几种方式，在 Windows 环境下，一般只需要在 Windows Explorer 中双击.exe 文件，但最好打开一个命令行窗口，输入执行它的命令，因为在程序执行完毕后，显示输出的窗口就会消失。在所有操作系统环境上，都可以从命令行运行程序。只需要启动一个命令行会话，把当前目录改为包含程序可执行文件的目录，再输入程序名，就可以执行它了。

如果没有出现错误，就大功告成了。这个程序会在屏幕上输出如下信息:

```
Hello world!
```

1.6 编辑第一个程序

我们可以修改程序，在屏幕上输出其他信息。例如可以将程序改成：

```
/*Program 1.2 Your Second C Program */
#include <stdio.h>

int main(void)
{
   printf("\"If at first you don't succeed, try, try, try again!\"");
   return 0;
```

```
}
```

这个版本的输出是：

```
"If  at  first  you  don't  succeed,  try,  try,  try  again! "
```

在要显示的文本中，\"序列称为转义序列(escape sequence)。文本中包含几个不同的转义序列。\"是在文本中包含双引号的特殊方式，因为双引号通常表示字符串的开头和结尾。转义序列\"使双引号出现在输出的开头和结尾。如果不使用转义序列，不仅双引号不会出现在输出中，而且程序不会被编译。本章后面的“控制字符”一节将详细介绍转义序列。

修改完源代码后，可以重新编译，链接后执行。反复练习，熟悉整个流程。

1.7 处理错误

犯错乃人之常情，没什么可难为情的。幸好计算机一般不会出错，而且非常擅长于找出我们犯的错误。编译器会列出在源代码中找到的一组错误信息(甚至比我们想象的多)，通常会指出有错误的语句。此时，我们必须返回编辑阶段，找出有错误的代码并更正。

有时一个错误会使后面本来正确的语句也出现错误。这多半是程序的其他部分引用了错误语句定义的内容所造成的。当然，定义语句有错，但被定义的内容不一定有错。

下面看看源代码在程序中生成了一个错误时会是什么样的情况。编辑第二个程序示例，将printf()行最后的分号去掉，如下所示：

```
/*Program 1.2 Your Second C Program */
#include <stdio.h)

int main(void)
{
  printf("\"If at first you don't succeed, try, try, try again!\"")
  return 0;
}
```

编译这个程序后，会看到错误信息，具体信息随编译器的不同而略有区别。下面是一个比较常见的错误信息：

```
Syntax error : expected ';' before 'return'
HELLO.C - 1 error(s), 0 warning(s)
```

编译器能精确地指出错误及其出处，在这里，printf()行的结尾处需要一个分号。在开始编写程序时，可能有很多错误是简单的拼写错误造成的。还很容易忘了逗号、括号，或按错了键。许多有经验的老手也常犯这种错误。

如前所述，有时一点小错误会造成大灾难，编译器会显示许多不同的错误信息。不要被错误的数量吓倒，仔细看过每一个错误信息后，返回并改掉错误部分，不懂的先不管它，然后再编译一次源文件，就会发现错误一次比一次少。

返回编辑器，重新输入分号，再编译，看看有没有其他错误。如果没有错误，程序就可以执行了。

1.8 剖析一个简单的程序

编写并编译了第一个程序后，下面是另一个非常类似的例子，了解各行代码的作用。

```
/* Program 1.3 Another Simple C Program - Displaying a Quotation */
#include  <stdio.h>

int  main(void)
{
  printf("Beware  the  Ides  of  March!");
  return 0;
}
```

这和第一个程序完全相同，这里把它作为练习，用编辑器输入这个示例，编译并执行。若输入完全正确，会看到如下输出：

```
Beware  the  Ides  of  March!
```

1.8.1 注释

上述示例的第一行代码如下：

```
/* Program 1.3 Another Simple C Program - Displaying a Quotation */
```

这不是程序代码，因为它没有告诉计算机执行操作。它只是一个注释，告诉阅读代码的人，这个程序要做什么。位于/*和*/之间的任意文本都是注释。只要编译器在源文件中找到/*，就忽略它后面的内容(即使其中的文本很像程序代码)，一直到表示注释结束的*/为止。/*可以和*/放在同一行代码上，也可以放在不同的代码行上。如果忘记包含对应的*/，编译器就会忽略/*后面的所有内容。下面使用一个注释说明代码的作者及版权。

```
/*
 * Written by Ivor Horton
 * Copyright 2012
*/
```

也可以修饰注释，使它们比较突出。

```
/ ******************************************
  * This is a very important comment         *
  * so please read this.                     *
 ******************************************/
```

使用另一种记号，可以在代码行的末尾添加一个注释，如下所示。

```
printf("Beware the Ides of March!");    // This line displays a quotation
```

代码行上两个斜杠后面的所有内容都会被编译器忽略。这种形式的注释没有前一种记号那么凌乱，尤其是在注释只占一行的情形下。

应养成给程序添加注释的习惯，当然程序也可以没有注释，但在编写较长的程序时，可能会忘记这个程序的作用或工作方式。添加足够的注释，可确保日后自己(和其他程序员)能理解程序的作用和工作方式。

下面再给程序添加一些注释。

```
/* Program 1.3 Another Simple C Program - Displaying a Quotation */
#include <stdio.h>                          // This is a preprocessor directive

int main(void)                              // This identifies the function main()
{                                           // This marks the beginning of main()
  printf("Beware the Ides of March!");      // This line outputs a quotation
  return 0;                                 // This returns control to the operating system
}                                           // This marks the end of main()
```

可以看出，使用注释是一种非常有效的方式，可以解释程序中要发生的事情。注释可以放在程序中的任意位置，说明代码的一般作用，指定代码是如何工作的。

1.8.2 预处理指令

下面的代码行：

```
#include <stdio.h>                  // This is a preprocessor directive
```

严格说来，它不是可执行程序的一部分，但它很重要，事实上程序没有它是不执行的。符号#表示这是一个预处理指令(preprocessing directive)，告诉编译器在编译源代码之前，要先执行一些操作。编译器在编译过程开始之前的预处理阶段处理这些指令。预处理指令相当多，大多放在程序源文件的开头。

在这个例子中，编译器要将 stdio.h 文件的内容包含进来，这个文件称为头文件(header file)，因为它通常放在程序的开头处。在本例中，头文件定义了 C 标准库中一些函数的信息，但一般情况下，头文件指定的信息应由编译器用于在程序中集成预定义函数或其他全局对象，所以有时需要创建自己的头文件，以用于程序。本例要用到标准库中的 printf()函数，所以必须包含 stdio.h 头文件。stdio.h 头文件包含了编译器理解 printf()以及其他输入/输出函数所需要的信息。名称 stdio 是标准输入/输出(standard input/output)的缩写。C 语言中所有头文件的扩展名都是.h，本书的后面会用到其他头文件。

■ **注意：**在一些系统中，头文件名是不区分大小写的，但在#include 指令里，这些文件名通常是小写。

每个符合 C11 标准的 C 编译器都有一些标准的头文件。这些头文件主要包含了与 C 标准库函数相关的声明。所有符合该标准的 C 编译器都支持同一组标准库函数，有同一组标准头文件，但一些编译器有额外的库函数，它们提供的功能一般是运行编译器的计算机所专用的。

■ **注意：**附录 E 列出了所有的标准头文件。

1.8.3 定义 main()函数

下面的 5 行指令定义了 main()函数。

```
int main(void)                              // This identifies the function main()
{                                           // This marks the beginning of main()
  printf("Beware the Ides of March!");      // This line outputs a quotation
  return 0;                                 // This returns control to the operating system
}                                           // This marks the end of main()
```

函数是两个括号之间执行某组操作的一段代码。每个 C 程序都由一个或多个函数组成，每个 C 程序都必须有一个 main()函数，因为每个程序总是从这个函数开始执行。因此，假定创建、编译、链接了一个名为 progname.exe 的文件。执行它时，操作系统会执行这个程序的 main()函数。

定义 main()函数的第一行代码如下：

```
int main(void)              // This identifies the function main()
```

它定义了 main()函数的起始。注意这行代码的末尾没有分号。定义 main()函数的第一行代码开头是一个关键字 int，它表示 main()函数的返回值的类型，关键字 int 表示 main()函数返回一个整数值。执行完 main()函数后返回的整数值表示返回给操作系统的一个代码，它表示程序的状态。在下面的语句中，指定了执行完 main()函数后要返回的值。

```
return 0;                  // This returns control to the operating system
```

这个 return 语句结束 main()函数的执行，把值 0 返回给操作系统。从 main()函数返回 0 表示，程序正常终止，而返回非 0 值表示异常。换言之，在程序结束时，发生了不应发生的事情。

紧跟在函数名 main 后的括号，带有函数 main()开始执行时传递给它的信息。在这个例子里，括号内是 void，表示没有给函数 main()传递任何数据，后面会介绍如何将数据传递给函数 main()或程序内的其他函数。

函数 main()可以调用其他函数，这些函数又可以调用其他函数。对于每个被调用的函数，都可以在函数名后面的括号中给函数传递一些信息。在执行到函数体中的 return 语句时，就停止执行该函数，将控制权返回给调用函数(对于函数 main()，则将控制权返回给操作系统)。一般函数会定义为有返回值或没有返回值。函数返回一个值时，该值总是特定的类型。对于函数 main()，返回值的类型是 int，即整数。

1.8.4 关键字

在 C 语言中，关键字是有特殊意义的词语，所以在程序中不能将关键字用于其他目的。关键字也称为保留字。在前面的例子里，int 就是一个关键字，void 和 return 也是关键字。C 语言有许多关键字，我们在学习 C 语言的过程中，将逐渐熟悉这些关键字。附录 C 列出了完整的 C 语言关键字表。

1.8.5 函数体

main()函数的一般结构如图 1-2 所示。

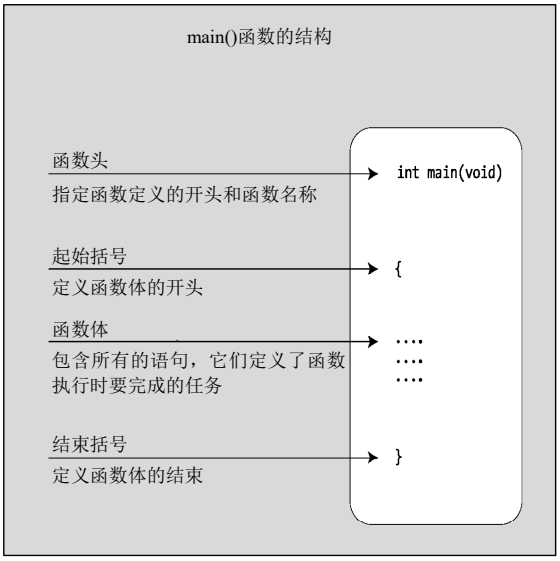

图 1-2　函数 main()的结构

函数体是在函数名称后面位于起始及结束两个大括号之间的代码块。它包含了定义函数功能的所有语句。这个例子的 main()函数体非常简单，只有两个语句：

```
{                                       // This marks the beginning of main()
  printf("Beware the Ides of March!");  // This line outputs a quotation
  return 0;                             // This returns control to the operating system
}                                       // This marks the end of main()
```

每个函数都必须有函数体，但函数体可以是空的，仅有起始及结束两个大括号，里面没有任何语句。这种情况下，这个函数什么也不做。

这样的函数有什么用？事实上，在开发一个包含很多函数的程序时，这种函数是非常有用的。我们可以声明一些用来解决手头问题的空函数，确定需要完成的编程工作，再为每个函数创建程序代码。这个方法有助于条理分明地、系统地建立程序。

■ **注意：**Program 1.3 将大括号单独排为一行，并缩进大括号之间的代码。这么做可清楚地表示括号框起来的语句块从哪里起始和结束。大括号之间的语句通常缩进两个或多个空格，使大括号突出在前。这是很好的编程格式，可以使语句块更容易阅读。

代码中的大括号可以用其他方式摆放。例如：

```
int main(void) {
  printf("Beware the Ides of March!");    // This line outputs a quotation
  return 0;
}
```

■ **提示：**无论源代码采用什么方式摆放，都要一直采用这种方式，这很重要。

1.8.6 输出信息

例子中的 main()函数体包含了一个调用 printf()函数的语句。

```
printf("Beware the Ides of March!");      // This line outputs a quotation
```

printf()是一个标准的库函数，它将函数名后面引号内的信息输出到命令行上(实际上是标准输出流，默认为命令行)。在这个例子中，调用这个函数会显示双引号内的一段警示语：双引号内的字符串称为字符串字面量。注意这行代码用分号作为结尾。

1.8.7 参数

包含在函数名(如上面语句中的 printf()函数)后的圆括号内的项称为参数，它指定要传送给函数的数据。当传送给函数的参数多于一个时，要用逗号分开。

在上面的例子中，函数的参数是双引号内的文本字符串。如果不喜欢例子中引号内的文本，可以改用自己想输出的句子。例如，使用如下语句：

```
printf("Out, damned Spot! Out I say!");
```

修改源代码后，必须再次编译及链接程序，才可执行。

■ **注意：**与 C 语言中所有可执行的语句一样，printf()行的末尾必须有分号(这与定义语句或指令语句不同)。这是一个很容易犯的错误，尤其是初次使用 C 编程的人，老是忘了分号。

1.8.8 控制符

前面的程序可以改为输出两段句子。输入以下的代码：

```
// Program 1.4 Another Simple C Program - Displaying a Quotation
#include <stdio.h>

int main(void)
{
  printf("My formula for success?\nRise early, work late, strike oil.\n");
  return 0;
}
```

输出的结果是：

```
My formula for success?
Rise early, work late, strike oil.
```

在 printf()语句中，在文本的开头和第一句的后面，增加了字符\n，它是另一个转义序列，代表换行符。这样输出光标就会移动到下一行，后续的输出就会显示在新行上。

反斜杠(\)在文本字符串里有特殊的意义，它表示转义序列的开始。反斜杠后面的字符表示是哪种转义序列。对于\n，n 表示换行。还有其他许多转义序列。显然，反斜杠是有特殊意义的，所以需要一种方式在字符串中指定反斜杠。为此，应使用两个反斜杠(\\)。

输入以下程序：

```
// Program 1.5 Another Simple C Program - Displaying Great Quotations
#include <stdio.h>

int main(void)
{
   printf("\"It is a wise father that knows his own child.\"\nShakespeare\n");
   return 0;
}
```

输出的结果如下：

```
"It is a wise father that knows his own child."
Shakespeare
```

输出中包含双引号，因为在字符串中使用了双引号的转义序列。Shakespeare 显示在下一行，因为在\"的后面有\n 转义序列。

在输出字符串中使用转义序列\a 可以发出声音，说明发生了有趣或重要的事情。输入以下的程序并执行：

```
// Program 1.6 A Simple C Program – Important
#include <stdio.h>

int main(void)
{
   printf("Be careful!!\n\a");
   return 0;
}
```

这个程序的输出如下所示且带有声音。仔细聆听，电脑的扬声器会发出鸣响。

```
Be  careful!!
```

转义序列\a 表示发出鸣响。表 1-1 是转义序列表。

表 1-1　转义序列

转义序列	说明
\n	换行
\r	回车键
\b	退后一格
\f	换页
\t	水平制表符
\v	垂直制表符
\a	发出鸣响
\?	插入问号(?)

(续表)

转义序列	说明
\"	插入双引号(")
\'	插入单引号(')
\\	插入反斜杠(\)

试着在屏幕上显示多行文本，在该文本中插入空格。使用 \n 可以把文本放在多行上，使用\t 可以给文本加上空格。本书将大量使用这些转义序列。

1.8.9 三字母序列

一般可以直接在字符串中使用问号。\?转义序列存在的唯一原因是，有 9 个特殊的字母序列，称为三字母序列，这是包含 3 个字母的序列，分别表示#、[,]、\、^、~、\、{和}。

??=转换为#	??(转换为[	??)转换为]
??/转换为\	??<转换为{	??>转换为}
??'转换为^	??!转换为|	??-转换为~

在 International Organization for Standardization(ISO)不变的代码集中编写 C 代码时，就需要它们，因为它没有这些字符。这可能不适用于你。可以完全不理会它们，除非希望编写如下语句：

```
printf("What??!\n");
```

这个语句生成的输出如下：

```
What|
```

三字母序列??!会转换为|。为了获得希望的输出，需要把上述语句写成：

```
printf("What?\?!\n");
```

现在三字母序列不会出现，因为第二个问号用其转义序列指定。使用三字母序列时，编译器会发出一个警告，因为通常是不应使用三字母序列的。在现代编译器(如 GCC)中，输出中将出现警告，以避免错误的三字母错误解释，也可以通过参数强制执行-Wtrigraphs：

```
program1_08.c:7:15: warning: trigraph ??! ignored, use -trigraphs to enable [-Wtrigraphs]
```

1.9 预处理器

上例介绍了如何使用预处理指令，把头文件的内容包含到源文件中。编译的预处理阶段可以做的工作不止于此。除了指令外，源文件还可以包含宏。宏是提供给预处理器的指令，来添加或修改程序中的 C 语句。宏可以很简单，只定义一个符号，例如 INCHES_PER_FOOT，只要出现这个符号，就用 12 替代。其指令如下：

```
#define INCHES_PER_FOOT 12
```

在源文件中包含这个指令，则代码中只要出现 INCHES_PER_FOOT，就用 12 替代它。例如：

```
printf("There are %d inches in a foot.\n", INCHES_PER_FOOT);
```

预处理后，这个语句变成：

```
printf("There are %d inches in a foot.\n", 12);
```

INCHES_PER_FOOT 不再出现，因为该符号被#define 指令中指定的字符串替代。对于源文件中的每个符号实例，都会执行这个替代。

宏也可以很复杂，根据特定的条件把大量代码添加到源文件中。这里不进一步介绍。第 13 章将详细讨论预处理器宏。在此之前我们会遇到一些宏，那时会解释它们。

1.10 用 C 语言开发程序

如果读者从未写过程序，对 C 语言开发程序的过程就不会很清楚，但它和我们日常生活的许多事务是相同的。万事开头难。一般首先大致确定要实现的目标，接着把该目标转变成比较准确的规范。有了这个规范后，就可以制订达到最终目标的一系列步骤了。就好比光知道要盖房子是不够的，还得知道需要盖什么样的房子，它有多大，用什么材料，要盖在哪里。这种详细规划也需要运用到编写程序上。下面介绍编写程序时需要完成的基本步骤。房子的比喻是很有帮助的，因此就利用这个比喻。

1.10.1 了解问题

第一步是弄清楚要做什么。在不清楚应提供什么设施——多少间卧房、多少间浴室、各房间多大等之前就开始建造房子，会有不知所措之感。所有这些都会影响建造房子所需的材料和工作量，从而影响整个房子的成本。一般来说，在满足需求和完成项目的有限资金、人力及时间之间总会达成某种一致。

这和开发一个任意规模的程序是相同的。即使是很简单的问题，也必须知道有什么输入，对输入该做什么处理，要输出什么，以及输出哪种格式。输入可以来自键盘，也可以来自磁盘文件的数据，或来自电话或网络的信息。输出可以显示在屏幕上，或打印出来，也可以是更新磁盘上的数据文件。

对于较复杂的程序，需要多了解程序的各个方面。清楚地定义程序要解决的问题，对于理解制订最终方案所需的资源与努力，是绝对必要的一部分。好好考虑这些细节，还可以确定项目是否切实可行。对于新项目缺乏精准、详细的规范，使项目所花的时间和资金大大超出预算因而中断项目的例子有很多。

1.10.2 详细设计

要建造房子，必须有详细的计划。这些计划能让建筑工人按图施工，并详细描述房子如何建造——具体的尺寸、要使用的材料等。还需要确定何时完成什么工作。例如，在砌墙之前要先挖地基，所以这个计划必须把工作分为可管理的单元，以便执行起来井然有序。

写程序也是一样。首先将程序分解成许多定义清楚且互相独立的小单元，描述这些独立单元相互沟通的方式，以及每个单元在执行时需要什么信息，从而开发出富有逻辑、相互独立的单元。把大型程序编写为一个大单元肯定是不可行的。

1.10.3 实施

有了房子的详细设计，就可以开始工作了。每组建筑工人必须按照进度完成他们的工作。在下一阶段开始前，必须先检查每个阶段是否正确完成。省略了这些检查，将可能导致整栋房子倒塌。

当然，假使程序很大，可以一次编写一部分。一个部分完成后，再写下一部分。每个部分都要基于详细的设计规范，在进行下一个部分之前，应尽可能详细地检查每个部分的功能。这样，程序就会逐步完成预期的任务。

大型编程项目常常涉及一组程序员。项目应分成相当独立的单元，分配给程序员组中的各个成员。这样就可以同时开发几个代码单元。如果代码单元要相互连接为一个整体，就必须精确定义代码单元与程序其余部分之间的交互。

1.10.4 测试

房子完成了，还要进行许多测试：排水设备、水电设施、暖气等。任何部分都有可能出问题，这些问题必须解决。这有时是一个反复的过程，一个地方的问题可能会造成其他地方出问题。

这个机制与写程序是类似的。每个程序模块——组成程序的单元——都需要单独测试。若它们工作不正常，就必须调试。调试是一个找出程序中的问题及更正错误的过程。调试的由来有个说法，曾经有人在查找程序的错误时，使用计算机的电路图来跟踪信息的来源及其处理方式，竟然发现计算机程序出现错误，是因为一只虫子在计算机里，让里面的线路短路而发生的，后来，bug 这个词就成了程序错误的代名词。

对于简单的程序，通常只要检查代码，就可以找出错误。然而一般来说，调试过程通常会使用调试器临时插入一些代码，确定在出错时会发生什么。这包括插入断点，应当暂停执行，检查代码中的值。还可以单步执行代码。如果没有调试器，就要加入额外的程序代码，输出一些信息，以确定程序中事件的发生顺序，以及程序执行时生成的中间值。在大型程序里，还需要联合测试各个程序模块，因为各个模块或许能正常工作，但并不保证它能和其他模块一起正常工作。在程序开发的这个阶段，有个专业术语称为集成测试。

1.11 函数及模块化编程

到目前为止，“函数”这个词已出现过好几次了，如 main()、printf()、函数体等。下面将深入研究函数是什么，为什么它们那么重要。

大多数编程语言(包含 C 语言)都提供了一种方法，将程序切割成多个段，各段都可以独立编写。在 C 语言中，这些段称为函数(尽管它们被称为函数，但这是一种命令式/过程性语言，而不是函数式语言)。一个函数的程序代码与其他函数是相互隔绝的。函数与外界有一个特殊的接口，可将信息传进来，也可将函数产生的结果传出去。这个接口在函数的第一行(即在函数名的地方)指定。

图 1-3 的简单程序例子由 4 个函数组成，用于分析棒球分数。

这 4 个函数都完成一个指定的、定义明确的工作。程序中操作的执行由一个模块 main()总体掌控。一个函数负责读入及检查输入数据，另一个函数进行分析。读入及分析了数据后，第 4 个函数就输出球队及球员的排名。

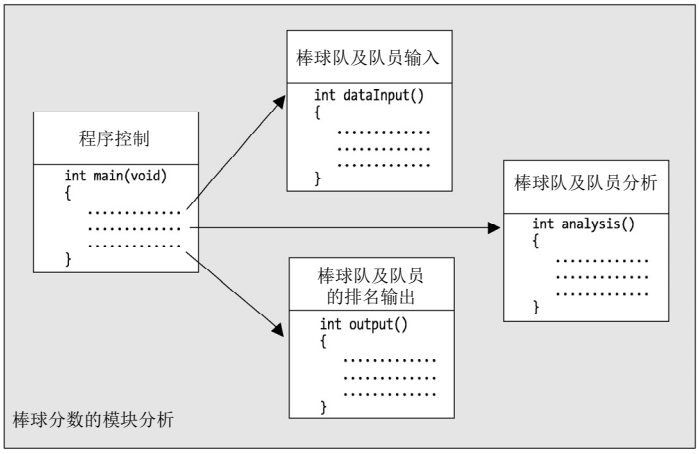

图 1-3　模块化编程

将程序分成多个易于管理的小单元，对编程是非常重要的，其理由如下。

- 单独编写和测试每个函数，可以大大简化使整个程序运转起来的过程。
- 几个独立的小函数比一个大函数更容易处理和理解。
- 库就是供人使用的函数集。因为它们是事先写好且经过测试的，能正常工作，所以可以放心地使用，无须细究它的代码细节。这就加快了开发程序的速度，因为我们只需要关注自己的代码，这是 C 语言的一个基本组成部分。C 语言中丰富的函数库大大增强了 C 语言的能力。
- 也可以编写自己的函数库，应用于自己感兴趣的程序类型。如果发现经常编写某个函数，就可以编写它的通用版本，以满足自己的需求，并将它加入自己的库中。以后需要用到这个函数时，就可使用它的库版本了。
- 在开发包含几千到几百万行代码的大型程序时，可以由一些程序设计团队来进行，每个团队负责一个指定的函数子组，最后把它们组成完整的程序。

第 8 章将详细介绍 C 函数。C 程序的结构在本质上就是函数的结构，本章的第一个例子就用到一个标准的库函数 printf()。

■ **注意：**在其他一些编程语言中，用术语“方法”表示自包含的代码单元。因此方法的含义与函数相同。

试试看：将所学的知识用于实践

下面的例子将前面学到的知识用于实践。首先，看看下面的代码，检查自己是否理解它的作用。然后输入这些代码，编译、链接并执行，看看会发生什么。

```
// Program 1.7 A longer program
#include <stdio.h>              // Include the header file for input and output

int main(void)
{
  printf("Hi there!\n\n\nThis program is a bit");
  printf(" longer than the others.");
```

```
  printf("\nBut really it's only more text.\n\n\n\a\a");
  printf("Hey, wait a minute!! What was that???\n\n");
  printf("\t1.\tA bird?\n");
  printf("\t2.\tA plane?\n");
  printf("\t3.\tA control character?\n");
  printf("\n\t\t\b\bAnd how will this look when it prints out?\n\n");
  return 0;
}
```

输出如下:

```
Hi there!

This program is a bit longer than the others.
But really it's only more text.

Hey, wait a minute!! What was that???

      1.    A bird?
      2.    A plane?
      3.    A control character?

And how will this look when it prints out?
```

代码的说明

这个程序看起来有点复杂，这只是因为括号内的文本字符串包含了许多转义序列。每个文本字符串都由一对双引号括起来。但这个程序只是连续调用 printf()函数，说明屏幕输出是由传送给 printf()函数的数据所控制。

本例通过预处理指令包含了标准库中的 stdio.h 文件。

```
#include <stdio.h>              // Include the header file for input and output
```

这是一个预处理指令，因为它以符号#开头。stdio.h 文件提供了使用 printf()函数所需的定义。然后，定义 main()函数头，指定它返回一个整数值。

```
int  main(void)
```

括号中的 void 关键字表示不给 main()函数传递信息。下一行的大括号表示其下是函数体:

```
{
```

下一行语句调用标准库函数 printf()，将“Hi there!”输出到屏幕上，接着空两行，输出“This program is a bit”。

```
printf("Hi there!\n\n\nThis  program  is  a  bit");
```

空两行是由 3 个转义序列\n 生成的。转义序列\n 会把字符显示在新行上。第一个转义序列\n 结束了包含“Hi there!”的行，之后的两个转义序列\n 生成两个空行，文本“This program is a bit”显示在第 4 行上。这行代码在屏幕上生成了 4 行输出。

下一个 printf()生成的输出跟在上一个 printf()输出的最后一个字符后面。下面的语句输出文本"longer than the others."，其中的第一个字符是一个空白。

```
printf(" longer  than  the  others.");
```

这个输出跟在上一个输出的后面，紧临 bit 中的 t。因此在文本的开头需要一个空格，否则计算机就会显示“This program is a bitlonger than the others. ”，这不是我们希望的结果。

下一个语句在输出前会先换行，因为双引号中文本字符串的开头是\n。

```
printf("\nBut  really  it's  only  more  text.\n\n\n\a\a");
```

显示完文本后会空两行(因为有 3 个\n 转义序列)，然后发出两次鸣响。下一个屏幕输出从空的第二行开始。

下一个输出语句如下。

```
printf("Hey, wait  a  minute!!  What  was  that???\n\n");
```

输出文本后空一行。其后的输出在空的这行开始。

以下 3 行语句各插入一个制表符，显示一个数字后，再插入另一个制表符，之后是一些文本，结束后换行。这样，输出更容易阅读。

```
printf("\t1.\tA  bird?\n");
printf("\t2.\tA  plane?\n");
printf("\t3.\tA  control  character?\n");
```

这几个语句会生成 3 行带编号的输出。

下一个语句先输出一个换行符，因此在前面输出的后面是一个空行，然后输出两个制表符和两个空格，接着退回两个空格，最后显示文本并换行。

```
printf("\n\t\t\b\bAnd how will this look when it prints out?\n\n");
```

函数体中的最后一个语句如下。

```
return 0;
```

这个语句结束 main()的执行，把 0 返回给操作系统。

结束大括号表示函数体结束。

```
}
```

■ **注意：**输出中制表符和退格的实际效果随编译器的不同而不同。

1.12 常见错误

错误是生活中的一部分。用 C 语言编写计算机程序时，必须用编译器将源代码转换成机器码，因此必须用非常严格的规则控制使用 C 语言的方式。漏掉一个该有的逗点，或添加不该有的分号，编译器都不会将程序转换成机器码。

即使实践了多年，程序中也很容易出现输入错误。这些错误可能在编译或链接程序时找出。但有些错误可能使程序执行时，表面上看起来正常，却不定时地出错，这就需要花很多时间跟踪错误了。

当然，不是只有输入错误会带来问题，具体实施时也常常会发现问题。在处理程序中复杂的判断结构时，很容易出现逻辑错误。从语言的观点看，程序是正确的，编译及运行也正确，但得不到正确的结果。这类错误最难查找。

1.13 要点

温习第一个程序是个不错的方法，图 1-4 列出了重点。

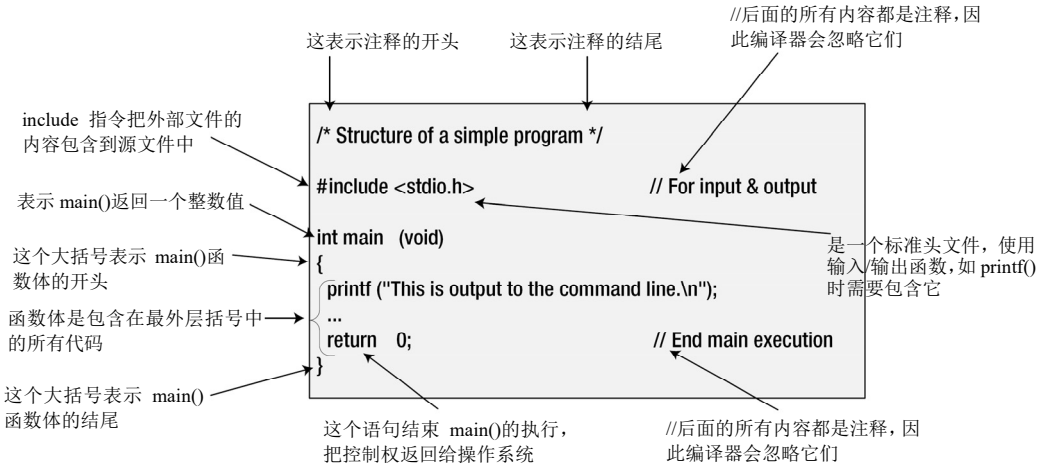

图 1-4 简单程序的要素

1.14 小结

本章编写了几个 C 程序。我们学习了许多基础知识，本章的重点是介绍一些基本概念，而不是详细探讨 C 程序语言。现在读者应该对编写、编译及链接程序很有信心了。也许读者目前对如何构建 C 程序只有模糊的概念。以后学了更多的 C 语言知识，编写了一些程序后，就会清楚明白了。

下一章将学习较复杂的内容，而不只是用 printf()输出文本。我们要处理信息，得到更有趣的结果。另外，printf()不只是显示文本字符串，它还有其他用途。

1.15 习题

以下的习题能让读者测试本章所学的成果。如果有不懂的地方，可以翻看本章的内容，还可以从 Apress 网站 www.apress.com 的 Source Code/Download 部分下载答案，但这应是最后一种方法。

习题 1.1 编写一个程序，用两个 printf()语句分别输出自己的名字及地址。

习题 1.2 将上一个练习改成所有的输出只用一个 printf()语句。

习题 1.3 编写一个程序，输出下列文本，格式如下所示。

```
"It's  freezing  in  here," he said coldly.
```