目录

C primer Plus Sixty Edition

- 经久不衰的 C 语言畅销经典教程
- 针对 C11 标准进行全面更新

美 |Stephen Prate| 著姜佑 | 译

1 前言

1984 年 C Primer Plus 第一版问世。各版本累计销售超 55 万册。C 语言从早期的非正式的 K&R 标准，发展到 1990 年 ISO/ANSI 标准，进而发展到 2011 年 ISO/IEC 标准。本书第六版。

1.1 目标:

- 指导性强、条理清晰而且有用的 C 语言教程。
- 一本友好、方便使用、便于自学的指南。
- 介绍 C 语言细节的同时，讲解编程的概念。假定读者为非专业的程序员。
- 当概念用文字较难解释时，以图表演示以助理解。
- C 语言的主要特征总结在方框中，便于查找及复习。
- 每章末有复习和编程练习，测试和加深

- 为有最佳学习效果，扮演一个积极的角色。仔细阅读，录入运行。
- * 手写与思考为最有效的方法，不要匆忙录入。*

2 第 1 章初识 C 语言

- C 的历史和特性
- 编写程序的步骤
- 编译器和链接器的知识
- C 标准

C 是一门功能强大、专业化的编程语言。

2.1 1.1C 语言的起源

1972 年，贝尔实验室的丹尼斯. 里厅 (Dennis Ritch) 和肯. 汤普逊 (Ken Thompso) 在开发 UNIX 操作系统时设计了 C 语言。Pascal 的主发目标是为更好地学习编程原理提供扎实的基础；而 BASIC 的主要目标是开发出类似英文的语言，让不熟悉计算机的人轻松学习。

2.2 1.2 选择 C 语言的理由

在过去 40 年里，C 成为最重要、最流行的语言之一。突出的优点

2.2.1 设计特性

C 是一门流行的语言，融合了计算机科学理论和实践的控制特性。C 语言的设计理念让用户能轻松地完成 * 自顶向下的规划、结构化编程和模块化设计。* 因此，用 C 语言编写的程序更 * 易懂、更可靠 *。

2.2.2 高效性

C 是高效的语言。在设计上，它充分利用了当前计算机的优势，因此 C 程序相对/更紧凑/，而且/运行速度很快/。实际上，C 语言具有通常是汇编语言才具有的微调控制能力 (汇编语言是为特殊的中央处理单元设计的一系列内部指令，使用助记符来表示; 不同的 CPU 系列使用不同的汇编语言)，可以根据具体情况微调程序以获得最大运行速度或最有效地使用内存。

2.2.3 可移植性

C 是可移植的语言。这意味着，在一种系统中编写的 C 程序稍作修改或不修改就能在其他系统运行。如需修改，也只需简单更改主程序头文件中的少许项即可。大部分语言都希望成为可移植语言，但是，如果经历过把 IBM PC BASIC 程序转换成苹果 BASIC(两者是近亲)，或者在 UNIX 系统中运行 IBM 大型机的 FORTRAN 程序的人都知道，移植是最麻烦的事。C 语言是可移植方面的佼佼者。从 8 位微处理器到克雷超级计算机，许多计算机体系结构都可以使用 C 编译器 (C 编译器是把 C 代码转换成计算机内部指令的程序)。但是要注意，程序中针对特殊硬件设备 (如，显示监视器) 或操作系统特殊功能 (如，Windows 8 或 OS X) 编写的部分，通常是不可移植的。由于 C 语言与 UNIX 关系密切，UNIX 系统通常会将 C 编译器作为软件包的一部分。安装 Linux 时，通常也会安装 C 编译器。供个人计算机使用的 C 编译器很多，运行各种版本的 Windows 和 Macintosh(即，Mac) 的 PC 都能找到合适的 C 编译器。因此，无论是使用家庭计算机、专业工作站，还是大型机，都能找到针对特定系统的 C 编译器。

2.2.4 强大而灵活

C 语言功能强大且灵活 (计算机领域经常使用这两个词)。例如，功能强大且灵活的 UNIX 操作系统，大部分是用 C 语言写的; 其他语言 (如，FORTRAN、Perl、Python、Pascal、LISP、Logo、BASIC) 的许多编译器和解释器都是用 C 语言编写的。因此，在 UNIX 机上使用 FORTRAN 时，最终是由 C 程序生成最后的可执行程序。C 程序可以用于解决物理学和工程学的问题，甚至可用于制作电影的动画特效。

2.2.5 面向程序员

C 语言是为了满足程序员的需求而设计的，程序员利用 C 可以访问硬件、操控内存中的位。C 语言有丰富的运算符，能让程序员简洁地表达自己的意图。C 没有 Pascal 严谨，但是却比 C++ 的限制多。这样的灵活性既是优点也是缺点。优点是，许多任务用 C 来处理都非常简洁 (如，转换数据的格式); 缺点是，你可能会犯一些莫名其妙的错误，这些错误不可能在其他语言中出现。C 语言在提供更多自由的同时，也让使用者承担了更大的责任。

另外，大多数 C 实现都有一个大型的库，包含众多有用的 C 函数。这些函数用于处理程序员经常需要解决的问题。

2.2.6 缺点

人无完人，金无足赤。C 语言也有一些缺点。例如，前面提到的，要享受用 C 语言自由编程的乐趣，就必须承担更多的责任。特别是，C 语言使用指针，而涉及指针的编程错误往往难以察觉。有句话说的好: 想拥有自由就必须时刻保持警惕。

C 语言紧凑简洁，结合了大量的运算符。正因如此，我们也可以编写出让人极其费解的代码。虽然没必要强迫自己编写晦涩的代码，但是有兴趣写写也无妨。试问，除 C 语言外还为哪种语言举办过年度混乱代码大赛 [1]?

瑕不掩瑜，C 语言的优点比缺点多很多。我们不想在这里多费笔墨，还是来聊聊 C 语言的其他话题。

2.3 1.3 C 语言的应用范围

早在 20 世纪 80 年代，C 语言就已经成为小型计算机 (UNIX 系统) 使用的主流语言。从那以后，C 语言的应用范围扩展到微型机 (个人计算机) 和大型机 (庞然大物)。如图 1.2 所示，许多软件公司都用 C 语言来开发文字处理程序、电子表格、编译器和其他产品，因为用 C 语言编写的程序紧凑而高效。更重要的是，C 程序很方便修改，而且移植到新型号的计算机中也没什么问题。

无论是软件公司、经验丰富的 C 程序员，还是其他用户，都能从 C 语言中受益。越来越多的计算机用户已转而求助 C 语言解决一些安全问题。不

一定非得是计算机专家也能使用 C 语言。

20 世纪 90 年代，许多软件公司开始改用 C++ 来开发大型的编程项目。C++ 在 C 语言的基础上嫁接了面向对象编程工具 (面向对象编程是一门哲学，它通过对语言建模来适应问题，而不是对问题建模以适应语言)。C++ 几乎是 C 的超集，这意味着任何 C 程序差不多就是一个 C++ 程序。学习 C 语言，也相当于学习了许多 C++ 的知识。

虽然这些年来 C++ 和 JAVA 非常流行，但是 C 语言仍是软件业中的核心技能。在最想具备的技能中，C 语言通常位居前十。特别是，C 语言已成为嵌入式系统编程的流行语言。也就是说，越来越多的汽车、照相机、DVD 播放机和其他现代化设备的微处理器都用 C 语言进行编程。除此之外，C 语言还从长期被 FORTRAN 独占的科学编程领域分得一杯羹。最终，作为开发操作系统的卓越语言，C 在 Linux 开发中扮演着极其重要的角色。因此，在进入 21 世纪的第 2 个 10 年中，C 语言仍然保持着强劲的势头。简而言之，C 语言是最重要的编程语言之一，将来也是如此。如果你想 33 拿下一份编程的工作，被问到是否会 C 语言时，最好回答 “是”。34

2.4 1.4 计算机能做什么

在学习如何用 C 语言编程之前，最好先了解一下计算机的工作原理。这些知识有助于你理解用 C 语言编写程序和运行 C 程序时所发生的事情之间有什么联系。现代的计算机由多种部件构成。中央处理单元 (CPU) 承担绝大部分的运算工作。随机存取内存 (RAM) 是存储程序和文件的工作区; 而永久内存存储设备 (过去一般指机械硬盘，现在还包括固态硬盘) 即使在关闭计算机后，也不会丢失之前储存的程序和文件。另外，还有各种外围设备 (如，键盘、鼠标、触摸屏、监视器) 提供人与计算机之间的交互。CPU 负责处理程序，接下来我们重点讨论它的工作原理。CPU 的工作非常简单，至少从以下简短的描述中看是这样。它从内存中获取并执行一条指令，然后再从内存中获取并执行下一条指令，诸如此类 (一个吉赫兹的 CPU 一秒钟能重复这样的操作大约十亿次，因此，CPU 能以惊人的速度从事枯燥的工作)。CPU 有自己的小工作区——由若干个寄存器组成，每个寄存器都可以储存一个数字。一个寄存器储存下一条指令的内存地址，CPU 使用该地址来获

取和更新下一条指令。在获取指令后，CPU 在另一个寄存器中储存该指令，并更新第 1 个寄存器储存下一条指令的地址。CPU 能理解的指令有限 (这些指令的集合叫作指令集)。而且，这些指令相当具体，其中的许多指令都是用于请求计算机把一个数字从一个位置移动到另一个位置。例如，从内存移动到寄存器。下面介绍两个有趣的知识。其一，储存在计算机中的所有内容都是数字。计算机以数字形式储存数字和字符 (如，在文本文档中使用的字母)。每个字符都有一个数字码。计算机载入寄存器的指令也以数字形式储存，指令集中的每条指令都有一个数字码。其二，计算机程序最终必须以数字指令码 (即，机器语言) 来表示。简而言之，计算机的工作原理是: 如果希望计算机做某些事，就必须为其提供特殊的指令列表 (程序)，确切地告诉计算机要做的事以及如何做。35

你必须用计算机能直接明白的语言 (机器语言) 创建程序。这是一项繁琐、乏味、费力的任务。计算机要完成诸如两数相加这样简单的事，就得分成类似以下几个步骤。1. 从内存位置 2000 上把一个数字拷贝到寄存器 1。2. 从内存位置 2004 上把另一个数字拷贝到寄存器 2。3. 把寄存器 2 中的内容与寄存器 1 中的内容相加，把结果储存在寄存器 1 中。4. 把寄存器 1 中的内容拷贝到内存位置 2008。而你要做的是，必须用数字码来表示以上的每个步骤! 如果以这种方式编写程序很合你的意，那不得不说抱歉，因为用机器语言编程的黄金时代已一去不复返。但是，如果你对有趣的事情比较感兴趣，不妨试试高级编程语言。36

2.5 1.5 高级计算机语言和编译器

高级编程语言 (如，C) 以多种方式简化了编程工作。首先，不必用数字码表示指令; 其次，使用的指令更贴近你如何想这个问题，而不是类似计算机那样繁琐的步骤。使用高级编程语言，可以在更抽象的层面表达你的想法，不用考虑 CPU 在完成任务时具体需要哪些步骤。例如，对于两数相加，可以这样写: total = mine + yours; 对我们而言，光看这行代码就知道要计算机做什么; 而看用机器语言写成的等价指令 (多条以数字码形式表现的指令) 则费劲得多。但是，对计算机而言却恰恰相反。在计算机看来，高级指令就是一堆无法理解的无用数据。编译器在这里派上了用场。编译器是把高级语

言程序翻译成计算机能理解的机器语言指令集的程序。程序员进行高级思维活动，而编译器则负责处理冗长乏味的细节工作。

编译器还有一个优势。一般而言，不同 CPU 制造商使用的指令系统和编码格式不同。例如，用 Intel Core i7 (英特尔酷睿 i7)CPU 编写的机器语言程序对于 ARM Cortex-A57 CPU 而言什么都不是。但是，可以找到与特定类型 CPU 匹配的编译器。因此，使用合适的编译器或编译器集，便可把一种高级语言程序转换成供各种不同类型 CPU 使用的机器语言程序。一旦解决了一个编程问题，便可让编译器集翻译成不同 CPU 使用的机器语言。简而言之，高级语言 (如 C、Java、Pascal) 以更抽象的方式描述行为，不受限于特定 CPU 或指令集。而且，高级语言简单易学，用高级语言编程比用机器语言编程容易得多。1964 年，控制数据公司 (Control Data Corporation) 研制出了 CDC 6600 计算机。这台庞然大物是世界上首台超级计算机，当时的售价是 600 万美元。它是高能核物理研究的首选。然而，现在的普通智能手机在计算能力和内存方面都超过它数百倍，而且能看视频，放音乐。37 1964 年，在工程和科学领域的主流编程语言是 FORTRAN。虽然编程语言不如硬件发展那么突飞猛进，但是也发生了很大变化。为了应对越来越大型的编程项目，语言先后为结构化编程和面向对象编程提供了更多的支持。随着时间的推移，不仅新语言层出不穷，而且现有语言也会发生变化。38

2.6 1.6 语言标准

目前，有许多 C 实现可用。在理想情况下，编写 C 程序时，假设该程序中未使用机器特定的编程技术，那么它的运行情况在任何实现中都应该相同。要在实践中做到这一点，不同的实现要遵循同一个标准。C 语言发展之初，并没有所谓的 C 标准。1987 年，布莱恩 · 柯林汉 (Brian Kernighan) 和丹尼斯 · 里奇 (Dennis Ritchie) 合著的 The C Programming Language(《C 语言程序设计》) 第 1 版是公认的 C 标准，通常称之为 K&R C 或经典 C。特别是，该书中的附录中的 “C 语言参考手册” 已成为实现 C 的指导标准。例如，编译器都声称提供完整的 K&R 实现。虽然这本书中的附录定义了 C 语言，但却没有定义 C 库。与大多数语言不同的是，C 语言比其他语言更依赖库，因此需要一个标准库。实际上，由于缺乏官方标准，UNIX 实现提

供的库已成为了标准库。

2.6.1 1.6.1 第 1 个 ANSI/ISO C 标准

随着 C 的不断发展，越来越广泛地应用于更多系统中，C 社区意识到需要一个更全面、更新颖、更严格的标准。鉴于此，美国国家标准协会 (ANSI) 于 1983 年组建了一个委员会 (X3J11)，开发了一套新标准，并于 1989 年正式公布。该标准 (ANSI C) 定义了 C 语言和 C 标准库。国际标准化组织于 1990 年采用了这套 C 标准 (ISO C)。ISO C 和 ANSI C 是完全相同的标准。ANSI/ISO 标准的最终版本通常叫作 C89(因为 ANSI 于 1989 年批准该标准) 或 C90(因为 ISO 于 1990 年批准该标准)。另外，由于 ANSI 先公布 C 标准，因此业界人士通常使用 ANSI C。在该委员会制定的指导原则中，最有趣的可能是: 保持 C 的精神。委员会在表述这一精神时列出了以下几点: 信任程序员; 不要妨碍程序员做需要做的事; 39

保持语言精练简单; 只提供一种方法执行一项操作; 让程序运行更快，即使不能保证其可移植性。在最后一点上，标准委员会的用意是: 作为实现，应该针对目标计算机来定义最合适的某特定操作，而不是强加一个抽象、统一的定义。在学习 C 语言过程中，许多方面都反映了这一哲学思想。

2.6.2 1.6.2 C99 标准

1994 年，ANSI/ISO 联合委员会 (C9X 委员会) 开始修订 C 标准，最终发布了 C99 标准。该委员会遵循了最初 C90 标准的原则，包括保持语言的精练简单。委员会的用意不是在 C 语言中添加新特性，而是为了达到新的目标。第 1 个目标是，支持国际化编程。例如，提供多种方法处理国际字符集。第 2 个目标是，“调整现有实践致力于解决明显的缺陷”。因此，在遇到需要将 C 移至 64 位处理器时，委员会根据现实生活中处理问题的经验来添加标准。第 3 个目标是，为适应科学和工程项目中的关键数值计算，提高 C 的适应性，让 C 比 FORTRAN 更有竞争力。这 3 点 (国际化、弥补缺陷和提高计算的实用性) 是主要的修订目标。在其他方面的改变则更为保守，例如，尽量与 C90、C++ 兼容，让语言在概念上保持简单。用委员会的话说: “委员会很满意让 C++ 成为大型、功能强大的语言”。C99 的修订

保留了 C 语言的精髓，C 仍是一门简洁高效的语言。本书指出了许多 C99 修改的地方。虽然该标准已发布了很长时间，但并非所有的编译器都完全实现 C99 的所有改动。因此，你可能发现 C99 的一些改动在自己的系统中不可用，或者只有改变编译器的设置才可用。

2.6.3 1.6.3 C11 标准维护标准任重道远。标准委员会在 2007 年承诺 C 标准的下一个版本是

40

C1X，2011 年终于发布了 C11 标准。此次，委员会提出了一些新的指导原则。出于对当前编程安全的担忧，不那么强调“信任程序员”目标了。而且，供应商并未像对 C90 那样很好地接受和支持 C99。这使得 C99 的一些特性成为 C11 的可选项。因为委员会认为，不应要求服务小型机市场的供应商支持其目标环境中用不到的特性。另外需要强调的是，修订标准的原因不是因为原标准不能用，而是需要跟进新的技术。例如，新标准添加了可选项支持当前使用多处理器的计算机。对于 C11 标准，我们浅尝辄止，深入分析这部分内容已超出本书讨论的范围。注意本书使用术语 ANSI C、ISO C 或 ANSI/ISO C 讲解 C89/90 和较新标准共有的特性，用 C99 或 C11 介绍新的特性。有时也使用 C90(例如，讨论一个特性被首次加入 C 语言时)。
41

2.7 1.7 使用 C 语言的 7 个步骤

C 是编译型语言。如果之前使用过编译型语言 (如，Pascal 或 FORTRAN)，就会很熟悉组建 C 程序的几个基本步骤。但是，如果以前使用的是解释型语言 (如，BASIC) 或面向图形界面语言 (如，Visual Basic)，或者甚至没接触过任何编程语言，就有必要学习如何编译。别担心，这并不复杂。首先，为了让读者对编程有大概的了解，我们把编写 C 程序的过程分解成 7 个步骤 (见图 1.3)。注意，这是理想状态。在实际的使用过程中，尤其是在较大型的项目中，可能要做一些重复的工作，根据下一个步骤的情况来调整或改进上一个步骤。图 1.3 编程的 7 个步骤

2.7.1 1.7.1 第 1 步: 定义程序的目标

42

在动手写程序之前，要在脑中有清晰的思路。想要程序去做什么首先自己要明确自己想做什么，思考你的程序需要哪些信息，要进行哪些计算和控制，以及程序应该要报告什么信息。在这一步骤中，不涉及具体的计算机语言，应该用一般术语来描述问题。

2.7.2 1.7.2 第 2 步: 设计程序

对程序应该完成什么任务有概念性的认识后，就应该考虑如何用程序来完成它。例如，用户界面应该是怎样的? 如何组织程序? 目标用户是谁? 准备花多长时间来完成这个程序? 除此之外，还要决定在程序 (还可能是辅助文件) 中如何表示数据，以及用什么方法处理数据。学习 C 语言之初，遇到的问题都很简单，没什么可选的。但是，随着要处理的情况越来越复杂，需要决策和考虑的方面也越来越多。通常，选择一个合适的方式表示信息可以更容易地设计程序和处理数据。再次强调，应该用一般术语来描述问题，而不是用具体的代码。但是，你的某些决策可能取决于语言的特性。例如，在数据表示方面，C 的程序员就比 Pascal 的程序员有更多选择。

2.7.3 1.7.3 第 3 步: 编写代码

设计好程序后，就可以编写代码来实现它。也就是说，把你设计的程序翻译成 C 语言。这里是真正需要使用 C 语言的地方。可以把思路写在纸上，但是最终还是要把代码输入计算机。这个过程的机制取决于编程环境，我们稍后会详细介绍一些常见的环境。一般而言，使用文本编辑器创建源代码文件。该文件中内容就是你翻译的 C 语言代码。程序清单 1.1 是一个 C 源代码的示例。程序清单 1.1 C 源代码示例 #include <stdio.h> 43

int main(void) { int dogs; printf("How many dogs do you have?"); scanf("%d", &dogs); printf("So you have %d dog(s)!", dogs); return 0; } 在这一步骤中，应该给自己编写的程序添加文字注释。最简单的方式是使用 C 的注释工具在源代码中加入对代码的解释。第 2 章将详细介绍如何在代码中添加注释。

2.7.4 1.7.4 第 4 步: 编译

接下来的这一步是编译源代码。再次提醒读者注意，编译的细节取决于编程的环境，我们稍后马上介绍一些常见的编程环境。现在，先从概念的角度讲解编译发生了什么事情。前面介绍过，编译器是把源代码转换成可执行代码的程序。可执行代码是用计算机的机器语言表示的代码。这种语言由数字码表示的指令组成。如前所述，不同的计算机使用不同的机器语言方案。C 编译器负责把 C 代码翻译成特定的机器语言。此外，C 编译器还将源代码与 C 库 (库中包含大量的标准函数供用户使用，如 printf() 和 scanf()) 的代码合并成最终的程序 (更精确地说，应该是由一个被称为链接器的程序来链接库函数，但是在大多数系统中，编译器运行链接器)。其结果是，生成一个用户可以运行的可执行文件，其中包含着计算机能理解的代码。44

编译器还会检查 C 语言程序是否有效。如果 C 编译器发现错误，就不生成可执行文件并报错。理解特定编译器报告的错误或警告信息是程序员要掌握的另一项技能。

2.7.5 1.7.5 第 5 步: 运行程序

传统上，可执行文件是可运行的程序。在常见环境 (包括 Windows 命令提示符模式、UNIX 终端模式和 Linux 终端模式) 中运行程序要输入可执行文件的文件名，而其他环境可能要运行命令 (如，在 VAX 中的 VMS[2]) 或一些其他机制。例如，在 Windows 和 Macintosh 提供的集成开发环境 (IDE) 中，用户可以在 IDE 中通过选择菜单中的选项或按下特殊键来编辑和执行 C 程序。最终生成的程序可通过单击或双击文件名或图标直接在操作系统中运行。

2.7.6 1.7.6 第 6 步: 测试和调试程序

程序能运行是个好迹象，但有时也可能会出现运行错误。接下来，应该检查程序是否按照你所设计的思路运行。你会发现你的程序中有一些错误，计算机行话叫作 bug。查找并修复程序错误的过程叫调试。学习的过程中不可避免会犯错，学习编程也是如此。因此，当你把所学的知识应用于编程时，最好为自己会犯错做好心理准备。随着你越来越老练，你所写的程序中的错

误也会越来越不易察觉。将来犯错的机会很多。你可能会犯基本的设计错误，可能错误地实现了一个好想法，可能忽视了输入检查导致程序瘫痪，可能会把圆括号放错地方，可能误用 C 语言或打错字，等等。把你将来犯错的地方列出来，这份错误列表应该会很长。看到这里你可能会有些绝望，但是情况没那么糟。现在的编译器会捕获许多错误，而且自己也可以找到编译器未发现的错误。在学习本书的过程中，我们会给读者提供一些调试的建议。

2.7.7 1.7.7 第 7 步: 维护和修改代码 45

创建完程序后，你发现程序有错，或者想扩展程序的用途，这时就要修改程序。例如，用户输入以 Zz 开头的姓名时程序出现错误、你想到了一个更好的解决方案、想添加一个更好的新特性，或者要修改程序使其能在不同的计算机系统中运行，等等。如果在编写程序时清楚地做了注释并采用了合理的设计方案，这些事情都很简单。

2.7.8 1.7.8 说明

编程并非像描述那样是一个线性的过程。有时，要在不同的步骤之间往复。例如，在写代码时发现之前的设计不切实际，或者想到了一个更好的解决方案，或者等程序运行后，想改变原来的设计思路。对程序做文字注释为今后的修改提供了方便。许多初学者经常忽略第 1 步和第 2 步 (定义程序目标和设计程序)，直接跳到第 3 步 (编写代码)。刚开始学习时，编写的程序非常简单，完全可以在脑中构思好整个过程。即使写错了，也很容易发现。但是，随着编写的程序越来越庞大、越来越复杂，动脑不动手可不行，而且程序中隐藏的错误也越来越难找。最终，那些跳过前两个步骤的人往往浪费了更多的时间，因为他们写出的程序难看、缺乏条理、让人难以理解。要编写的程序越大越复杂，事先定义和设计程序环节的工作量就越大。磨刀不误砍柴工，应该养成先规划再动手编写代码的好习惯，用纸和笔记录下程序的目标和设计框架。这样在编写代码的过程中会更加得心应手、条理清晰。46

2.8 1.8 编程机制

生成程序的具体过程因计算机环境而异。C 是可移植性语言，因此可以在许多环境中使用，包括 UNIX、Linux、MS-DOS(一些人仍在使用)、Windows 和 Macintosh OS。有些产品会随着时间的推移发生演变或被取代，本书无法涵盖所有环境。首先，来看看许多 C 环境 (包括上面提到的 5 种环境) 共有的一些方面。虽然不必详细了解计算机内部如何运行 C 程序，但是，了解一下编程机制不仅能丰富编程相关的背景知识，还有助于理解为何要经过一些特殊的步骤才能得到 C 程序。用 C 语言编写程序时，编写的内容被储存在文本文件中，该文件被称为源代码文件 (source code file)。大部分 C 系统，包括之前提到的，都要求文件名以.c 结尾 (如，wordcount.c 和 budget.c)。在文件名中，点号 (.) 前面的部分称为基本名 (basename)，点号后面的部分称为扩展名 (extension)。因此，budget 是基本名，c 是扩展名。基本名与扩展名的组合 (budget.c) 就是文件名。文件名应该满足特定计算机操作系统的特殊要求。例如，MS-DOS 是 IBM PC 及其兼容机的操作系统，比较老旧，它要求基本名不能超过 8 个字符。因此，刚才提到的文件名 wordcount.c 就是无效的 DOS 文件名。有些 UNIX 系统限制整个文件名 (包括扩展名) 不超过 14 个字符，而有些 UNIX 系统则允许使用更长的文件名，最多 255 个字符。Linux、Windows 和 Macintosh OS 都允许使用长文件名。接下来，我们来看一下具体的应用，假设有一个名为 concrete.c 的源文件，其中的 C 源代码如程序清单 1.2 所示。程序清单 1.2 c 程序 #include <stdio.h> int main(void) 47

{ printf("Concrete contains gravel and cement."); return 0; } 如果看不懂程序清单 1.2 中的代码，不用担心，我们将在第 2 章学习相关知识。

2.8.1 1.8.1 目标代码文件、可执行文件和库

C 编程的基本策略是，用程序把源代码文件转换为可执行文件 (其中包含可直接运行的机器语言代码)。典型的 C 实现通过编译和链接两个步骤来完成这一过程。编译器把源代码转换成中间代码，链接器把中间代码和其他代码合并，生成可执行文件。C 使用这种分而治之的方法方便对程序进行模块化，可以独立编译单独的模块，稍后再用链接器合并已编译的模块。通过

这种方式，如果只更改某个模块，不必因此重新编译其他模块。另外，链接器还将你编写的程序和预编译的库代码合并。中间文件有多种形式。我们在这里描述的是最普遍的一种形式，即把源代码转换为机器语言代码，并把结果放在目标代码文件 (或简称目标文件) 中 (这里假设源代码只有一个文件)。虽然目标文件中包含机器语言代码，但是并不能直接运行该文件。因为目标文件中储存的是编译器翻译的源代码，这还不是一个完整的程序。目标代码文件缺失启动代码 (startup code)。启动代码充当着程序和操作系统之间的接口。例如，可以在 MS Windows 或 Linux 系统下运行 IBM PC 兼容机。这两种情况所使用的硬件相同，所以目标代码相同，但是 Windows 和 Linux 所需的启动代码不同，因为这些系统处理程序的方式不同。目标代码还缺少库函数。几乎所有的 C 程序都要使用 C 标准库中的函数。例如，concrete.c 中就使用了 printf() 函数。目标代码文件并不包含该函 48

数的代码，它只包含了使用 printf() 函数的指令。printf() 函数真正的代码储存在另一个被称为库的文件中。库文件中有许多函数的目标代码。链接器的作用是，把你编写的目标代码、系统的标准启动代码和库代码这 3 部分合并成一个文件，即可执行文件。对于库代码，链接器只会把程序中要用到的库函数代码提取出来 (见图 1.4)。图 1.4 编译器和链接器简而言之，目标文件和可执行文件都由机器语言指令组成的。然而，目标文件中只包含编译器为你编写的代码翻译的机器语言代码，可执行文件中还包含你编写的程序中使用的库函数和启动代码的机器代码。49

在有些系统中，必须分别运行编译程序和链接程序，而在另一些系统中，编译器会自动启动链接器，用户只需给出编译命令即可。接下来，了解一些具体的系统。

2.8.2 1.8.2 UNIX 系统

由于 C 语言因 UNIX 系统而生，也因此而流行，所以我们从 UNIX 系统开始 (注意: 我们提到的 UNIX 还包含其他系统，如 FreeBSD，它是 UNIX 的一个分支，但是由于法律原因不使用该名称)。

1. 1. 在 UNIX 系统上编辑 UNIX C 没有自己的编辑器，但是可以使用通用的 UNIX 编辑器，如 emacs、jove、vi 或 X Window System 文本

编辑器。作为程序员，要负责输入正确的程序和为储存该程序的文件起一个合适的文件名。如前所述，文件名应该以.c 结尾。注意，UNIX 区分大小写。因此，budget.c、BUDGET.c 和 Budget.c 是 3 个不同但都有效的 C 源文件名。但是 BUDGET.C 是无效文件名，因为该名称的扩展名使用了大写 C 而不是小写 c。假设我们在 vi 编译器中编写了下面的程序，并将其储存在 inform.c 文件中: #include <stdio.h> int main(void) { printf("A .c is used to end a C program filename."); return 0; } 50

以上文本就是源代码，inform.c 是源文件。注意，源文件是整个编译过程的开始，不是结束。

2. 2. 在 UNIX 系统上编译 虽然在我们看来，程序完美无缺，但是对计算机而言，这是一堆乱码。计算机不明白 #include 和 printf 是什么 (也许你现在也不明白，但是学到后面就会明白，而计算机却不会)。如前所述，我们需要编译器将我们编写的代码 (源代码) 翻译成计算机能看懂的代码 (机器代码)。最后生成的可执行文件中包含计算机要完成任务所需的所有机器代码。以前，UNIX C 编译器要调用语言定义的 cc 命令。但是，它没有跟上标准发展的脚步，已经退出了历史舞台。但是，UNIX 系统提供的 C 编译器通常来自一些其他源，然后以 cc 命令作为编译器的别名。因此，虽然在不同的系统中会调用不同的编译器，但用户仍可以继续使用相同的命令。编译 inform.c，要输入以下命令: cc inform.c 几秒钟后，会返回 UNIX 的提示，告诉用户任务已完成。如果程序编写错误，你可能会看到警告或错误消息，但我们先假设编写的程序完全正确 (如果编译器报告 void 的错误，说明你的系统未更新成 ANSI C 编译器，只需删除 void 即可)。如果使用 ls 命令列出文件，会发现有一个 a.out 文件 (见图 1.5)。该文件是包含已翻译 (或已编译) 程序的可执行文件。要运行该文件，只需输入: a.out 输出内容如下: A .c is used to end a C program filename. 51

图 1.5 用 UNIX 准备 C 程序如果要储存可执行文件 (a.out)，应该把它重命名。否则，该文件会被下一次编译程序时生成的新 a.out 文件替换。如何处理目标代码?C 编译器会创建一个与源代码基本名相同的目

标代码文件，但是其扩展名是.o。在该例中，目标代码文件是 inform.o。然而，却找不到这个文件，因为一旦链接器生成了完整的可执行程序，就会将其删除。如果原始程序有多个源代码文件，则保留目标代码文件。学到后面多文 52

件程序时，你会明白到这样做的好处。

2.8.3 1.8.3 GNU 编译器集合和 LLVM 项目

GNU 项目始于 1987 年，是一个开发大量免费 UNIX 软件的集合 (GNU 的意思是 “GNU’ s Not UNIX”，即 GNU 不是 UNIX)。GNU 编译器集合 (也被称为 GCC，其中包含 GCC C 编译器) 是该项目的产品之一。GCC 在一个指导委员会的带领下，持续不断地开发，它的 C 编译器紧跟 C 标准的改动。GCC 有各种版本以适应不同的硬件平台和操作系统，包括 UNIX、Linux 和 Windows。用 gcc 命令便可调用 GCC C 编译器。许多使用 gcc 的系统都用 cc 作为 gcc 的别名。LLVM 项目成为 cc 的另一个替代品。该项目是与编译器相关的开源软件集合，始于伊利诺伊大学的 2000 份研究项目。它的 Clang 编译器处理 C 代码，可以通过 clang 调用。有多种版本供不同的平台使用，包括 Linux。2012 年，Clang 成为 FreeBSD 的默认 C 编译器。Clang 也对最新的 C 标准支持得很好。GNU 和 LLVM 都可以使用-v 选项来显示版本信息，因此各系统都使用 cc 别名来代替 gcc 或 clang 命令。以下组合: cc -v 显示你所使用的编译器及其版本。gcc 和 clang 命令都可以根据不同的版本选择运行时选项来调用不同 C 标准。gcc -std=c99 inform.c[3] gcc -std=c1x inform.c gcc -std=c11 inform.c 53

第 1 行调用 C99 标准，第 2 行调用 GCC 接受 C11 之前的草案标准，第 3 行调用 GCC 接受的 C11 标准版本。Clang 编译器在这一点上用法与 GCC 相同。

2.8.4 1.8.4 Linux 系统

Linux 是一个开源、流行、类似于 UNIX 的操作系统，可在不同平台 (包括 PC 和 Mac) 上运行。在 Linux 中准备 C 程序与在 UNIX 系统中几乎一样，不同的是要使用 GNU 提供的 GCC 公共域 C 编译器。编译命令类似

于: gcc inform.c 注意，在安装 Linux 时，可选择是否安装 GCC。如果之前没有安装 GCC，则必须安装。通常，安装过程会将 cc 作为 gcc 的别名，因此可以在命令行中使用 cc 来代替 gcc。欲详细了解 GCC 和最新发布的版本，请访问 http://www.gnu.org/software/gcc/index.html。

2.8.5 1.8.5 PC 的命令行编译器

C 编译器不是标准 Windows 软件包的一部分，因此需要从别处获取并安装 C 编译器。可以从互联网免费下载 Cygwin 和 MinGW，这样便可在 PC 上通过命令行使用 GCC 编译器。Cygwin 在自己的视窗运行，模仿 Linux 命令行环境，有一行命令提示。MinGW 在 Windows 的命令提示模式中运行。这和 GCC 的最新版本一样，支持 C99 和 C11 最新的一些功能。Borland 的 C++ 编译器 5.5 也可以免费下载，支持 C90。源代码文件应该是文本文件，不是字处理器文件 (字处理器文件包含许多额外的信息，如字体和格式等)。因此，要使用文本编辑器 (如，Windows Notepad) 来编辑源代码。如果使用字处理器，要以文本模式另存文件。源代码文件的扩展名应该是.c。一些字处理器会为文本文件自动添加.txt 扩展名。如果出现这种情况，要更改文件名，把 txt 替换成 c。通常，C 编译器生成的中间目标代码文件的扩展名是.obj(也可能是其 54

他扩展名)。与 UNIX 编译器不同，这些编译器在完成编译后通常不会删除这些中间文件。有些编译器生成带.asm 扩展名的汇编语言文件，而有些编译器则使用自己特有的格式。一些编译器在编译后会自动运行链接器，另一些要求用户手动运行链接器。在可执行文件中链接的结果是，在原始的源代码基本名后面加上.exe 扩展名。例如，编译和链接 concrete.c 源代码文件，生成的是 concrete.exe 文件。可以在命令行输入基本名来运行该程序:
C>concrete

2.8.6 1.8.6 集成开发环境 (Windows)

许多供应商 (包括微软、Embarcadero、Digital Mars) 都提供 Windows 下的集成开发环境，或称为 IDE(目前，大多数 IDE 都是 C 和 C++ 结合的编译器)。可以免费下载的 IDE 有 Microsoft Visual Studio Express 和 Pelles

C。利用集成开发环境可以快速开发 C 程序。关键是，这些 IDE 都内置了用于编写 C 程序的编辑器。这类集成开发环境都提供了各种菜单 (如，命名、保存源代码文件、编译程序、运行程序等)，用户不用离开 IDE 就能顺利编写、编译和运行程序。如果编译器发现错误，会返回编辑器中，标出有错误的行号，并简单描述情况。初次接触 Windows IDE 可能会望而生畏，因为它提供了多种目标 (target)，即运行程序的多种环境。例如，IDE 提供了 32 位 Windows 程序、64 位 Windows 程序、动态链接库文件 (DLL) 等。许多目标都涉及 Windows 图形界面。要管理这些 (及其他) 选择，通常要先创建一个项目 (project)，以便稍后在其中添加待使用的源代码文件名。不同的产品具体步骤不同。一般而言，首先使用【文件】菜单或【项目】菜单创建一个项目。选择正确的项目形式非常重要。本书中的例子都是一般示例，针对在简单的命令行环境中运行而设计。Windows IDE 提供多种选择以满足用户的不同需求。例如，Microsoft Visual Studio 提供【Win32 控制台应用程序】选项。对于其他系统，查找一个诸如【DOS EXE】、【Console】或 55

【Character Mode】的可执行选项。选择这些模式后，将在一个类控制台窗口中运行可执行程序。选择好正确的项目类型后，使用 IDE 的菜单打开一个新的源代码文件。对于大多数产品而言，使用【文件】菜单就能完成。你可能需要其他步骤将源文件添加到项目中。通常，Windows IDE 既可处理 C 也可处理 C++，因此要指定待处理的程序是 C 还是 C++。有些产品用项目类型来区分两者，有些产品 (如，Microsoft Visual C++) 用.c 文件扩展名来指明使用 C 而不是 C++。当然，大多数 C 程序也可以作为 C++ 程序运行。欲了解 C 和 C++ 的区别，请参阅参考资料 IX。你可能会遇到一个问题: 在程序执行完毕后，执行程序的窗口立即消失。如果不希望出现这种情况，可以让程序暂停，直到按下 Enter 键，窗口才消失。要实现这种效果，可以在程序的最后 (return 这行代码之前) 添加下面一行代码: getchar(); 该行读取一次键的按下，所以程序在用户按下 Enter 键之前会暂停。有时根据程序的需要，可能还需要一个击键等待。这种情况下，必须用两次 getchar(): getchar(); getchar(); 例如，程序在最后提示用户输入体重。用户键入体重后，按下 Enter 键以输入数据。程序将读取体重，第 1 个 getchar() 读取 Enter 键，第 2 个 getchar() 会导致程序暂停，直至用户再次按下 Enter 键。如果

你现在不知所云，没关系，在学完 C 输出后就会明白。到时，我们会提醒读者使用这种方法。虽然许多 IDE 在使用上大体一致，但是细节上有所不同。就一个产品的系列而言，不同版本也是如此。要经过一段时间的实践，才会熟悉编译器的工作方式。必要时，还需阅读使用手册或网上教程。56

Microsoft Visual Studio 和 C 标准在 Windows 软件开发中，Microsoft Visual Studio 及其免费版本 Microsoft Visual Studio Express 都久负盛名，它们与 C 标准的关系也很重要。然而，微软鼓励程序员从 C 转向 C++ 和 C#。虽然 Visual Studio 支持 C89/90，但是到目前为止，它只选择性地支持那些在 C++ 新特性中能找到的 C 标准 (如，long long 类型)。而且，自 2012 版本起，Visual Studio 不再把 C 作为项目类型的选项。尽管如此，本书中的绝大多数程序仍可用 Visual Studio 来编译。在新建项目时，选择 C++ 选项，然后选择【Win32 控制台应用程序】，在应用设置中选择【空项目】。几乎所有的 C 程序都能与 C++ 程序兼容。所以，本书中的绝大多数 C 程序都可作为 C++ 程序运行。或者，在选择 C++ 选项后，将默认的源文件扩展名.cpp 替换成.c，编译器便会使用 C 语言的规则代替 C++。

2.8.7 1.8.7 Windows/Linux

许多 Linux 发行版都可以安装在 Windows 系统中，以创建双系统。一些存储器会为 Linux 系统预留空间，以便可以启动 Windows 或 Linux。可以在 Windows 系统中运行 Linux 程序，或在 Linux 系统中运行 Windows 程序。不能通过 Windows 系统访问 Linux 文件，但是可以通过 Linux 系统访问 Windows 文档。

2.8.8 1.8.8 Macintosh 中的 C 目前，苹果免费提供 Xcode 开发系统下载 (过去，它有时免费，有时付

费)。它允许用户选择不同的编程语言，包括 C 语言。Xcode 凭借可处理多种编程语言的能力，可用于多平台，开发超大型的项目。但是，首先要学会如何编写简单的 C 程序。在 Xcode 4.6 中，通过【File】菜单选择【New Project】，然后选择【OS X Application Command Line Tool】，接着输入产品名并选择 C 类型。Xcode 使用 Clang 或 GCC C 编译器来编译 C 代码，

它以前默认使用 GCC，但是现在默认使用 Clang。可以设置选择使用哪一个编译器和哪一套 C 标准 (因为许可方面的事宜，Xcode 中 Clang 的版本比 GCC 的版本要新)。57

UNIX 系统内置 Mac OS X，终端工具打开的窗口是让用户在 UNIX 命令行环境中运行程序。苹果在标准软件包中不提供命令行编译器，但是，如果下载了 Xcode，还可以下载可选的命令行工具，这样就可以使用 clang 和 gcc 命令在命令行模式中编译。

2.9 1.9 本书的组织结构

本书采用多种方式编排内容，其中最直接的方法是介绍 A 主题的所有内容、介绍 B 主题的所有内容，等等。这对参考类书籍来说尤为重要，读者可以在同一处找到与主题相关的所有内容。但是，这通常不是学习的最佳顺序。例如，如果在开始学习英语时，先学完所有的名词，那你的表达能力一定很有限。虽然可以指着物品说出名称，但是，如果稍微学习一些名词、动词、形容词等，再学习一些造句规则，那么你的表达能力一定会大幅提高。为了让读者更好地吸收知识，本书采用螺旋式方法，先在前几个章节中介绍一些主题，在后面章节再详细讨论相关内容。例如，对学习 C 语言而言，理解函数至关重要。因此，我们在前几个章节中安排一些与函数相关的内容，等读者学到第 9 章时，已对函数有所了解，学习使用函数会更加容易。与此类似，前几章还概述了一些字符串和循环的内容。这样，读者在完全弄懂这些内容之前，就可以在自己的程序中使用这些有用的工具。59

2.10 1.10 本书的约定

在学习 C 语言之前，先介绍一下本书的格式。1.10.1 字体本书用类似在屏幕上或打印输出时的字体 (一种等宽字体)，表示文本程序和计算机输入、输出。前面已经出现了多次，如果读者没有注意到，字体如下所示: #include <stdio.h> int main(void) { printf("Concrete contains gravel and cement."); return 0; } 在涉及与代码相关的术语时，也使用相同的等宽字体，如 stdio.h。本书用等宽斜体表示占位符，可以用具体的项替换这些占位符。例如，下面是一个声明的模型: $\text{type}_{\text{name}}$ $\text{variable}_{\text{name}}$; 这里，可用 int 替换 $\text{type}_{\text{name}}$，

用 $\text{zebra}_{\text{count}}$ 替换 $\text{variable}_{\text{name}}$。1.10.2 程序输出本书用相同的字体表示计算机的输出，粗体表示用户输入。例如，下面是第 14 章中一个程序的输出: 60

Please enter the book title. Press [enter] at the start of a line to stop. My Life as a Budgie Now enter the author. Mack Zackle s 如上所示，以标准计算机字体显示的行表示程序的输出，粗体行表示用户的输入。可以通过多种方式与计算机交互。在这里，我们假设读者使用键盘键入内容，在屏幕上阅读计算机的响应。1. 特殊的击键通常，通过按下标有 Enter、c/r、Return 或一些其他文字的键来发送指令。本书将这些按键统一称为 Enter 键。一般情况下，我们默认你在每行输入的末尾都会按下 Enter 键。尽管如此，为了标示一些特定的位置，本书使用 [enter] 显式标出 Enter 键。方括号表示按下一次 Enter 键，而不是输入 enter。除此之外，书中还会提到控制字符 (如，Ctrl+D)。这种写法的意思是，在按下 Ctrl 键 (也可能是 Control 键) 的同时按下 D 键。2. 本书使用的系统 C 语言的某些方面 (如，储存数字的空间大小) 因系统而异。本书在示例中提到“我们的系统”时，通常是指在 iMac 上运行 OS X 10.8.4，使用 Xcode 4.6.2 开发系统的 Clang 3.2 编译器。本书的大部分程序都能使用 Windows7 系统的 Microsoft Visual Studio Express 2012 和 Pelles C 7.0，以及 Ubuntu13.04 Linux 系统的 GCC 4.7.3 进行编译。3. 读者的系统 61

你需要一个 C 编译器或访问一个 C 编译器。C 程序可以在多种计算机系统中运行，因此你的选择面很广。确保你使用的 C 编译器与当前使用的计算机系统匹配。本书中，除了某些示例要求编译器支持 C99 或 C11 标准，其余大部分示例都可在 C90 编译器中运行。如果你使用的编译器是早于 ANSI/ISO 的老式编译器，在编译时肯定要经常调整，很不方便。与其如此，不如换个新的编译器。大部分编译器供应商都为学生和教学人员提供特惠版本，详情请查看供应商的网站。1.10.3 特殊元素本书包含一些强调特定知识点的特殊元素，提示、注意、警告，将以如下形式出现在本书中: 边栏 边栏提供更深入的讨论或额外的背景，有助于解释当前的主题。提示 提示一般都短小精悍，帮助读者理解一些特殊的编程情况。警告 用于警告读者注意一些潜在的陷阱。注意 提供一些评论，提醒读者不要误入歧途。62

2.11 1.11 本章小结

C 是强大而简洁的编程语言。它之所以流行，在于自身提供大量的实用编程工具，能很好地控制硬件。而且，与大多数其他程序相比，C 程序更容易从一个系统移植到另一个系统。C 是编译型语言。C 编译器和链接器是把 C 语言源代码转换成可执行代码的程序。用 C 语言编程可能费力、困难，让你感到沮丧，但是它也可以激发你的兴趣，让你兴奋、满意。我们希望你在愉快的学习过程中爱上 C。63

2.12 1.12 复习题

复习题的参考答案在附录 A 中。1. 对编程而言，可移植性意味着什么？2. 解释源代码文件、目标代码文件和可执行文件有什么区别？3. 编程的 7 个主要步骤是什么？4. 编译器的任务是什么？5. 链接器的任务是什么？64

2.13 1.13 编程练习我们尚未要求你编写 C 代码，该练习侧重于编程过程的早期步骤。

2.13.1 1. 你刚被 MacroMuscle 有限公司聘用。该公司准备进入欧洲市场，需要一个把英寸单位转换为厘米单位 (1 英寸 =2.54 厘米) 的程序。该程序要提示用户输入英寸值。你的任务是定义程序目标和设计程序 (编程过程的第 1 步和第 2 步)。

[1]. 国际 C 语言混乱代码大赛 (IOCCC，The International Obfuscated C Code Contest)。这是一项国际编程赛事，从 1984 年开始，每年举办一次 (1997、1999、2002、2003 和 2006 年除外)，目的是写出最有创意且最让人难以理解的 C 语言代码。——译者注 [2].VAX(Virtual Address eXtension) 是一种可支持机器语言和虚拟地址的 32 位小型计算机。VMS(Virtual Memory System) 是旧名，现在叫 OpenVMS，是一种用于服务器的操作系统，可在 VAX、Alpha 或 Itanium 处理器系列平台上运行。——译者注 [3].GCC 最基本的用法是:gcc [options] [filenames]，其中 options 是所需的参数，filenames 是文件名。——译者注 65

3 第 2 章 C 语言概述

本章介绍以下内容：

- 运算符：=
- 函数：main()、printf()
- 编写一个简单的 C 程序
- 创建整形变量，为其赋值并在屏幕上显示其值
- 换行字符
- 如何在程序中写注释，创建包含多个函数的程序，发现程序的错误
- 什么是关键字

C 程序是什么样子的？浏览本书，能看到许多示例。初见 C 程序会觉得有些古怪，程序中的许多 {、cp-tor 和 ptr++ 这样的符号。然而，在学习 C 的过程中，对这些符号和 C 语言特有的其他符号会越来越熟悉，甚至会喜欢上它们。如果熟悉与 C 相关的其他语言，会对 C 语言有似曾相识的感觉。本章，我们从演示一个简单的程序示例开始，解释该程序的功能。同时，强调一些 C 语言的基本特性。

3.1 2.1 简单的 C 程序示例

我们来看一个简单的 C 程序，如程序清单 2.1 所示。该程序演示了用 C 语言编程的一些基本特性。请先通读程序清单 2.1，看看自己是否能明白该程序的用途，再认真阅读后面的解释。

程序清单 2.1 first.c 程序

```
// This is 2.1 first.c
//first
//
//Created by Mac on<2020-04-16 Thu>
//Copyright @ 2020 Mac All rights reserved.
```

```
        //
        #include<stdio.h>
        int main(void)  /*一个简单的C程序*/
      {
      int num;  /*定义一个名为num的变量*/
      num=1;  /*为num赋一个值*/
      printf("I am a simple");  /*使用printf()函数*/
      printf("computer.\n");
printf("My favorite number is %d because it is first.\n",num);
return ;
    }
```

输出内容为:

I am a simple computer.

My favorite number is 1 because it is first.

程序调整

程序的输出是否在屏幕上一闪而过? 某些窗口环境会在单独的窗口运行程序，然后在程序运行结束后自动关闭窗口。如果遇到这种情况，可以在程序中加代码，让窗口等待用户按下一个键后才关闭。一种方法是，在程序的 return 语句前加一行代码:

getchar();

这行代码让程序等待击键，窗口会在用户按下一个键后才关闭。在第八章详细介绍 getchar() 的内容。

3.2 2.2 示例解释

把程序清单 2.1 分析两遍。第 1 遍（快速概要）概述程序中每行代码的作用，帮助读者初步了解程序。第 2 遍（程序细节）详细分析代码的具体含义，帮助读者深入理解程序。

图 2.1 总结了组成 C 程序的几个部分，图中包含的元素比第 1 个程序多

如何插入图:

函数中 C 的 6 种语句:

- 标号语句
- 复合语句
- 表达式语句
- 选择语句
- 迭代语句
- 跳转语句

C 语言语句的构成:

- 关键字
- 标识符
- 运算符
- 数据

3.2.1 2.2.1 第 1 遍：快速概要

本节简述程序中的每行代码的作用。下一节详细讨论代码的含义。

#include<stdio.h> 包含另一个文件

该行告诉编译器把 stdio.h 中的内容包含在当前程序中。stdio.h 是 C 编译器软件包的标准部分，它提供键盘输入和屏幕输出的支持。

int main(void) 函数名

C 程序包含一个或多个函数，它们是 Ctkcyr 基本模块。程序清单 2.1 的程序中有一个名为 main() 的函数。圆括号表明 main() 是一个函数名。int 表明 main() 函数返回一个整数，void 表明 main() 不带任何参数。int and void 是标准 ANSI C 定义 main() 的一部分 () 如果使用 ANSI C 之前的编译器，可省略 void; 考虑兼容的问题，使用较新的 C 编译器。

一个简单的 *C* 程序 注释

注释在/* 和 */两个符号之间，这些注释能提高程序的可读性。注释只是帮助读者理解程序，编译器会忽略它们。

{ 函数体开始

左花括号表示函数定义开始，右花括号（}）表示函数定义结束。

int num; 声明

该声明表明，将使用一个名为 num 的变量，而且 num 是 int(整形) 类型。

num=1; 赋值表达式语句

语句 num=1; 把值 1 赋给名为 num 的变量。

printf("I am a simple"); 调用一个函数

该语句使用 printf() 函数，在屏幕上显示 I am a simple，光标停在同一行。printf() 是标准的 C 库函数。在程序中使用函数叫作 * 调用函数 *。

printf("computer.\"); 调用另一个函数

接下来调用的这个 printf() 函数在上条语句打印出来的内容后面加上"computer"。代码告诉计算机另起一行，即把光标移至下一行。

printf("My favorite number is %d because it is first.",num);

最后调用的 printf() 把 num 的值（1）内嵌在用双引号括起来的内容中一并打印。%d 告诉计算机以何种形式输出 num 的值，打印在何处。

return 0; return 语句

C 函数可以给调用方提供（或返回）一个数。目前，可暂时把该行看作是结束 main() 函数的要求。

} 结束

必须以右花括号表示程序结束。

3.2.2 2.2.2 第 2 遍：程序细节

浏览完程序清单 2.2 后，仔细分析程序。再次强调，将逐行分析程序中的代码，以每行代码为出发点，深入分析代码背后的细节，为更全面的学习 C 语言编程的特性夯实基础。

1. 1.#include 指令和头文件

   #include<stdio.h>

   这是程序的第一行。#include<stdio.h> 的作用相当于把 stdio.h 文件中的所有内容都输入该行所在的位置。实际上，这是一种 “拷贝-粘贴” 的操作。

   include 文件提供了一种方便的途径共享许多程序共有的信息。

   #include 这行代码是一条 C 预处理器指令 (proprocessor directive)。通常，C 编译器在编译 # 前会对源代码做一些准备工作，即预处理 (proprocessing)。

   所有的 C 编译器软件都提供 stdio.h 文件。该文件中包含了供编译器使用的输入和输出函数（如，printf()）信息。该文件名的含义是标准输入/输出头文件。通常，在 C 程序顶部的信息集合被称为头文件 (header)。

在大多数情况下，头文件包含了编译器创建最终可执行程序要用到的信息。例如，头文件中可以定义一些常量，或者指明函数名以及如何使用它们。但是，函数的实际代码在一个预编译代码的库文件中。简而言之，头文件帮助编译器把你的程序正确的组合在一起。

ANSI/ISO C 规定了 C 编译器必须提供哪些头文件。有些程序要包含 stdio.h，而有些不用。特定 C 实现的文档中应该包含对 C 库函数的说明。这些说明确定了使用哪些函数需要包含哪些头文件。例如，要使用 printf() 函数，必须包含 stdio.h 头文件。省略必要的头文件可能不会影响某一特定程序，但是最好不要这样做。本书每次用到库函数，都会用 #include 指令包含 ANSI/ISO 标准指定的头文件。

注意为何不内置输入和输出

读者一定很好奇，为何不把输入和输出这些基本功能内置在语言中。原因之一是，并非所有的程序都会用到 I/O（输入/输出）包。轻装上阵表现了 C 语言的哲学。正是这种经济使用资源的原则，使得 C 语言成为流行的嵌入式编程语言（例如，编写控制汽车自动燃油系统或蓝光播放机芯片的代码）。#include 中的 # 符号表明，C 预处理器在编译器接手之前处理这条指令。本书后面章节中会介绍更多预处理器指令的示例，第 16 章将更详细地讨论相关内容。

2. 2.main() 函数

   int main(void);

   程序清单 2.2 中的第 2 行表明该函数名为 main。的确，main 是一个极其普通的名称，但是这是唯一的选择。C 程序一定从 main() 函数开始执行（目前不必考虑例外的情况）。坠子 main() 函数，你可以任意命名其他函数，而且 main() 函数必须是开始的函数。圆括号有什么功能？用于识别 main() 是一个函数。很快你将学到更多的函数。目前而言，只需记住函数是 C 语序的基本模块。

   int 是 main() 函数的返回类型。表明 main() 函数返回的值是整数。返回到哪里？返回给操作系统。我们将在第 6 章再来探讨这个问题。

通常，函数名后面的圆括号中包含一些传入函数的信息。该例中没有传递任何信息。因此，圆括号内是单词 void（第 11 章次介绍把信息从 main() 函数传回操作系统的另一种形式）。

如果浏览旧式的 C 代码，会发现程序以如下形式开始：

main()

C90 标准勉强接受这种形式，但是 C99 和 C11 标准不允许这样写。因此，即使你使用的编译器允许，也不要这样写。

你还会看到下面这种形式：

void main()

一些编译器允许这样写，但是所有的标准都未认可这种写法。因此，编译器不必接受这种形式，而且许多编译器都不能这样写。需要强调的是，只要坚持使用标准形式，把程序从一个编译器移至另一个编译器时就不会出什么问题。

3. 3. 注释

一个简单的程序

在程序中，被/**/两个符号括起来的部分是程序的注释。写注释能让他人（包括自己）更容易明白你所写的程序。C 语言注释的好处之一是，可将注释放在任意的地方，甚至是与要解释的内容在同一行。较长的注释可单独放一行或多行。在/* 和 */之间的内容都会被编译器忽略。下面列出了一些有效和无效的注释形式：

这是一条 *C* 注释。

/* 这也是一条注释，

被分成了两行。*/

/*

也可以这样写注释。

*/

/* 这条注释无效，因为缺少了结束标记。

C99 新增了另一种风格的注释，普遍用于 C++ 和 Jave。这种新风格使用//符号创建注释，仅限于单行。

//这种注释只能写成一行。

int rigue;//这种注释也可置于此。

因为一行末尾就标志着注释的结束，所以这种风格的注释只需在注释开始下标明//符号即可。

这种新形式的注释是为了解决旧形式注释存在的潜在问题。假设有下面的代码：

```
/*
希望能运行
*/
x=100;
Y=200;
/* 其它内容已省略。 */
```

接下来，假设你决定删除第 4 行，但不小心删掉了第 3 行 (*/)。代码如下所示：

```
/*
希望能运行。
y=200;
/*其他内容已省略。*/
```

现在，编译器把第 1 行的/* 和第 4 行的 */配对，导致 4 行代码全部成了注释（包括应作为代码的那一行）。而//形式的注释只对单行有效，不会导致这种“消失代码”的问题。

一些编译器可能不支持这一特性。还有一些编译器需要更改设置，才能支持 C99 或 C11 的特性。

考虑到只有一种注释风格过于死板乏味，本书在示例中采用两种风格的注释。

4. 4. 花括号、函数体和块

{

...

}

程序清单 2.1 中，花括号把 main() 函数括起来。一般而言，所有的 C 函数都使用花括号标记函数体的开始和结束。这是规定，不能省略。只有花括号（{}）能起这种作用，圆括号（()）的方括号（[]）都不行。

花括号还可用于把函数中的多条语句合并为一个单元或块。如果读者熟悉 Pascal、ADA、Modula-2 或者 Algol，就会明白花括号在 C 语言中的作用类似于这些语言中的 begin 和 end。

5. 5. 声明

int num;

程序清单 2.1 中，这行代码叫作声明 (declaration)。声明是 C 语言最重要的特性之一。在该例中，声明完成了两件事。

其一，在函数中有一个名为 num 的变量（variable）。

其二，int 表明 num 是一个整数（即没有小数点或小数部分的数）。int 是一种数据类型。编译器使用这些信息为 num 变量在内存中分配存贮空间。分号在 C 语言中是大部分语句和声明的平分，不像在 Pascal 中只是语句间的分隔符。

int 是 C 语言的一个关键字（keyword），表示一种基本的 C 语言数据类型。关键字是语言定义的单词，不能做其他用途。例如，不能用 int 作为函数名我变量名。但是，这些关键字在该语言外不起作用，所以把一只猫或一个可爱的小孩叫 int 是可以的（尽管某些地方的当地习俗或法律可能不允许）。

示例中的 num 是一个标识符 (indentifier)，也就一个变量、函数或其他实体的名称。因此，声明把特定标识符与计算机内存中的特定位置联系起来，同时也确定了储存在某位置的信息类型或数据类型。

在 C 语言中，所有变量都必须先声明才能使用。这章法着必须列出程序中用到的所有变量名及其类型。

以前的 C 语言，还要求把变量声明在块的顶部，其他语句不能在任何声明的前面。也就是说，main() 函数体如下所示：

```
int main()//旧规则
{
int doors;
int dogs;
doors=5;
dogs=3;
//其他语句

}
```

C99 和 C11 遵循 C++ 的惯例，可以把声明放在块中任何位置。尽管如此，首次使用变量之前一定要先声明它。因此，如果编译器支持这一新特性，可以这样编写上面的代码：

```
int main()    //目前的C规则
{
//一些语句
int doors;
doors=5;//第1次使用 doors
```

```
//其他语句
int dogs;
dogs=3;//第1次使用 dogs
//其他语句
}
```

为了与旧系统更好地兼容，本书沿用最初的规则（即把变量声明都写在块的顶部）。

现在，读者可能有 3 问题：

- 什么是数据类型?
- 如何命名?
- 为何要声明变量?

(a) 数据类型

C 语言可以处理多种类型的数据，如整数、字符和浮点数。把变量声明为整形或字符类型，计算机才能正确地储存、读取和解释数据。下一章将详细介绍 C 语言中的各种数据类型。

(b) 命名

给变量命名时要使用有意义的变量或标识符（如，程序中需要一个变量数狗，该变量名应该是 $\text{dog}_{\text{count}}$ 而不是 x3）。如果变量名无法清楚地表达自身的用途，可在注释中进一步说明。这是一种良好的编程习惯和编程技巧。

C99 和 C11 允许使用更长的标识符名，但是编译器中识别前 63 年字符。对于外部标识符（参阅第 12 章），只允许使用 31 个字符。(以前 C90 只允许 6 个字符)，这是一个很大的进步。旧式编译器通常最多只允许使用 8 个字符)。实际上，你可以使用更长的字符，但是编译器会忽略超出的字符。也就是说，如果有两个标识符名都有 63 个字符，只有一个字符不同，那么编译器会识别这是两个不同的名称。如果两个标识符都是 64 个字符，只有

最后一个字符不同，那么编译器可能交其视为同一个名称，也可能不会。标准并未定义在这种情况下会发生什么。

可以用小写字母、大写字母、数字和下划线（——）来命名。而且，名称的第 1 个字符必须是字符或下划线，不能是数字。表 2.1 给出了一些示例。表 2.1 有效和无效的名称

有效的名称	无效的名称	原因
wiggles	$2]**	首字不为字母
cat2	2cat	
Hot_{Tub}	Hot-Tub	仅可用下划线
taxRate	tax rate	不能用空格
_kcab	don't	' 不能用

大小写字母

数字

下划线

首字不能为数字

C99 及 C11 长度 63 个字符

外部标识符长度为 31 个

C90 只允许 8 个字符

操作系统及 C 库标识符首字符默认下划线

操作系统和 C 库经常使用以一个或两个下划线字符开始的标识答（如，$_{\text{kcab}}$），因此最好避免在自己的程序中使用这种名称。标准标签都以一个或两个下划线字符开始，如库标识符。这样的标识符都是保留的。这意味着，虽然使用它们没有语法错误，但是会导致名称冲突。

C 语言的名称区分大小写，即把一个字母的大写和小写视为两个不同的字符。因此，stars 和 Stars、STARS 都不同。

为了让 C 语言更加国际化，C99 和 C11 根据通用字符名（即 UCN）机制添加了扩展字符集。其中包函了除英文字母以外的部分字符。欲了解详细内容，请参阅附录 B 的 “参考资料 VII：扩展字符支持”。

(c) 声明变量的四个理由

一些更老的语言（如，fortran 和 basic 的最初形式）都允许直接使用变量，不必先声明。为何 c 语言不采用这种简单易行的方法？原因如下。

把所有变量放在一处，方便读者查找和理解程序的用途。如果变量名都是有意义的（如，taxtate 而不是 r），这样做效果很好。如果变更名无法表述清楚，在注释中解释变量的含义。这种方法让程序的可读性更高。

声明变量会促使在编写程序之前做一些计划。程序在开始时要获得哪些信息？希望程序如何输出？表示数据最好的方式是什么？

声明变量有助于发现隐藏在程序中的小错误，如变量名拼写错误。例如，假设在某些不需要声明就可以直接使用变量的语言中，编写如下语句：

RADIUS1=20.R;

在后面的程序中，误写成：

CIRCUM=6.28*RADIUSl;

你不小心把数字 1 打成小写字母 l。这些语言会创建一个新的变量 RADIUSl，并使用该变量中的值（也许是 0，也许是垃圾值），导致赋给 CIRCUM 的值是错误值。可能要花很长时间才能查出原因。这种错误在 c 语言中不会发生（除非你很不明智地声明了两个极其相似的变量），因为编译器在发现未声明的 RADIUS1 时会报错。

如果事先未声明变量，c 程序将无法通过编译。如果前几个理由还不足以说服你，这个理由总可以让你认真考虑一下了。

如果要声明变量，应该声明在何处？前面提到过，c99 之前的标准要求氢声明都置于块的顶部，这样规定的好处是：把声明放在一起更容易理解程序的用途。c99 允许在需要时才声明变量，这样做的好处是：在给变量赋值之前声明变量，就不会忘记给变量赋值。但是实际上，许多编译器都不支持 c99。

6. 6. 赋值

   num=1;

   程序清单中的这行代码是赋值表达式语句。赋值是 c 语言的基本操作之一。该行代码的意思是“反值 1 赋给变量 num”。在执行 int num; 声明时，编译器在计算机内存中为变量 num 预留了空间，然后在执行这行赋值表达式语句时，把值储存在之前预留的位置。可以给 num 赋不同的值，这 num 之所以被称为变量（variable）的原因。注意，该赋值表达式语句从右侧把值赋到左侧。另外，该语句以分号结尾，如图 2.2 所示。

   其中等号为赋值运算符

7. 7.printf() 函数

   printf("I am a simple");

   printf("computer.");

   printf("My favorite number is %d because it is first.",num);

   这三行都使用了 C 语言的一个标准函数：printf()。圆括号表明 printf 是一个函数名。圆括号内的内容是从 main() 函数传递给 printf() 函数的信息。例如，上面的第 1 行把 I am a simple 传递给 printf() 函数。该信息被称为参数，或者更确切地说，是函数的实际参数（actual argument），【在 C 语言中，实际参数（简称实参）是传递给函数的特定值，形式参数（简称形参）是函数中用于储存值的变量。第 5 章中将详述相关内容。】printf() 函数用参数来做什么？该函数会查看双引号中的内容，并将其打印在屏幕上。

   第 1 行 printf() 演示了在 C 语言中如何调用函数。只需输入函数名，把所需的参数填入圆括号即可。当程序运行到这一行时，控制权被转给命名的函数（该例中是 printf()）。函数执行结束后，控制权被返回至主调函数（calling function），该例中是 main()。

   第 2 行 printf() 函数的双绰号中的字符并未输出。这是为什么？的意思是换行。组合（依次输入这两个字符）代表一个换行符（newline

charactor)。对于 printf() 而言，这的意思是 “在下一行的最左加开始新的一行”。也就是说，打印换行符的效果与在键盘按下 Enter 键相同。既然如此，为何不在键入 printf() 参数时直接使用 Enter 键? 因为编辑器可能认为这是直接的命令，而不是储存在源代码中的指令。换名话说，如果直接按下 Enter 键，编辑器会退出当前行并开始新的一行。但是，换行符公会影响程序输出的显示格式。

换行符是一个转义序列（escape sequence)。转义序列用于代表难以表示或无法输入的字符。如，表 Tab 键，代表 Backspace 键（退格键)。每个转义序列都以反斜杠字符 () 开始。我们在第 3 章再来探讨相关内容。

这样，就解释了为什么 3 行 printf() 语句只打印出两行：第 1 个 printf() 打印的内容中不含换行符，但是第 2 和第 3 个 printf() 中都有换行符。

第 3 个 printf() 还有一些不明之处：参数中的%d 在打印时有什么作用？先来看该函数的输出：

My favorite number is 1 because it is first.

对比发现，参数中的%d 被数字 1 代替了，而 1 就是变量 num 的值。%d 相当于是一个占位符，其作用是指明输出 num 值的位置。该行和下面的 BASIC 语句很像：

PRINT"My favorite number is ";num;because it is first."

实际上，C 语言的 printf() 比 BASIC 的这条语句做的事情多一些。% 提醒程序，要在该处打印一个变更，d 表明把变量作为十进制整数打印。printf() 函数名中的 f 提醒用户，这是一种格式化打印函数。printf() 函数有多种打印变量的格式，包括小数和十六进制整数。后面章节在介绍数据类型时，会详细介绍相关内容。

8. 8.return 语句

   return 0;

   return 语句是程序清单 2.1 的最后一条语句。int main(void) 中的 int 表明 main() 函数应返回一个整数。C 标准要求 main() 这样做。有返

回值的 C 函数要有 return 语句。该语句以 return 关键字开始，后面是待返回的值，并以分号结尾。如果遗漏 main() 函数中的 return 语句，程序在运行至最外面的右花括号 (}) 时会返回 0。因此，可以省略 main() 函数末尾的 return 语句。但是，不要在其他有返回值的函数中漏掉它。因此，强烈建议读者养成在 main() 函数中保留 return 语句的好习惯。在这种情况下，可将其看作是统一代码风格。但对于某些操作系统（包括 Linux 和 UNIX），return 语句有实际的用途。第 11 章再详述这个主题。

3.3 2.3 简单程序的结构

分析过一个具体的程序示例后，了解一个程序的基本结构。程序由一个或多个函数组成，必须有 main() 函数。函数由函数头和函数体组成。函数头包括函数名、传入该函数的信息类型和函数的返回类型。通过函数名后的圆括号可识别出函数，圆括号里可能为空，可能有参数。函数体被花括号括起来，由一系列语句、声明组成，如图 2.4 所示。本章的程序示例中有一条声明，声明了程序使用的变量名和类型。然后是一条赋值表达式语句，变量被赋给一个值。接下来是 3 条 printf() 语句，调用 printf() 函数 3 次。最后，main() 以 return 语句结束。

```
int main(void)//函数头
{
int num;//声明
num=1;//语句
printf("%d is neat. \n",num);//语句
return 0;//语句
}
```

图 2.4 函数包含函数头和函数体

阐而言之，一个简单的 C 程序的格式如下：

```
#include<stdio.h>
```

```
int main(void)
{
语句
return 0;
}
```

(大部分语句都以分号结尾。)

3.4 2.4 提高程序可读性的技巧

编写可读性高和程序是良好的编程习惯。可读性高的程序更容易理解，以后也更容易修改和更正。抽调程序的可读性还有助于理清编程思路。

前面介绍过两种提高程序可读性的技巧：选择有意义的函数名和写注释。注意，使用这两种技巧时应相得益彰，避免重复🅵嗦。如果变量名是 width, 就不必写注释说明该变量表示宽度，但是如果变量名是 $\text{video}_{\text{routine4}}$，就要解释一下该变量名的含义。

提高程序可读性的第 3 个技巧是：在函数中用空行分隔概念上的多个部分。例如，程序清单 2.1 中用窄把声明部分和程序的其他部分区分开来。C 语言并未规定一定要使用空行，但是多使用空行能提高程序的可读性。

提高程序可读性的第 4 个技巧是：每条语句各占一行。同样，这也不是 C 语言的要求。C 语言的格式比较自由，可以把多条语句放在一行，也可以每条语句独占一行。下面的语句都没问题，但是不好看：

```
int main(void){int four; four
=
4;
printf(
"%d\n",
four);return 0;}
```

分号告诉编译器一条语句在哪里结束、下一条语句在哪里开始。如果按照本章示例的约定来编写代码（见图 2.5），程序的逻辑会更清晰。

```
int main(void)/*把2音寻（测水深的单位）转换成英尺*/——写注释
```

```
{
int feet,fathoms;//使用有意义的变量名
                 //使用空行
fathoms=2;
feet=6*fathoms;  //每行一条语句
printf("There are %d feet in %d fathoms!\n",feet,fathoms);
return 0;
}
```

图 2.5 提高程序的可读性

3.5 2.5 进一步使用 C

本章的第 1 年程序相当简单，下面的程序清单 2.2 也不太难。

程序清单 2.2 $\text{fathm}_{\text{ft.c}}$ 程序

```
//fathm_ft.c --把2音寻转换成英寸

#include<stdio.h>
int main(void)
{
int feet,fathoms;
fathoms=2;
feet=6*fathoms;
printf("There are %d feet in %d fathoms!\n",feet,fathoms);
printf("Yes, I said%d feet!\n",6*fathoms);
return 0;
}
```

与程序清单 2.1 相比，以上代码有什么新内容？这段代码提供了程序描述，声明了多个变量，进行了乘法运算，并打印了两个变量的值。下面更详细地分析这些内容。

3.5.1 2.5.1 程序说明

程序在开始处有一条注释（使用新的注释风格），给出了文件名和程序的目的。写这种程序说明很简单、不费时，而且在以后浏览或打印程序时很有帮助。

3.5.2 2.5.2 多条声明

接下来，程序在一条声明中声明了两个变量，而不是一个变量。为此，要在声明中用逗号隔开两个变量（feet 和 fathoms）。也就是说，

int feet,fathoms;

和

int feet;

int fathoms;

等价。

3.5.3 2.5.3 乘法

然后，程序进行了乘法运算。利用计算机强大的计算能力来计算 6 乘以 2。C 语言和许多其他语言一样，用 * 表示乘法。因此，语句

feet=6*fathoms;

的意思是 “查找变量 fathoms 的值，用 6 乘以该值，并把计算结果赋给变量 feet”。

3.5.4 2.5.4 打印多个值

最后，程序以新的方式使用 printf() 函数。如果编译并运行该程序，输出应该是这样：

There are 12 feet in 2 fathoms!

Yes, I said 12 feet!

程序的第 1 个 printf() 中进行了两次替换。双引号后面的第 1 个变量（feet）替换了双引号中的第 1 个%d；双绰号后面的第 2 个变量（fathoms）

替换了双引号中的第 2 个%d。注意，待输出的变量列于双引号的后面。还要注意，变量之间要用逗号隔开。

第 2 个 printf() 函数说明待打印的值不一定是变量，只要可求值得出合适类型值的项即可，如 6*fathoms。

该程序涉及的范围有限，但它是把章寻转换成英寸程序的核心部分。还需要把其他值通过交互的方式赋给 feet，其方法将在后面章节中介绍。

3.6 2.6 多个函数

目前为止，介绍的几个程序只使用了 printf() 函数。程序清单 2.3 演示了除 main() 以外，如何把自己的函数加入程序中。

程序清单 2.3 $\text{two}_{\text{func.c}}$ 程序

```
//two_func.c ——一个文件中包含两个函数
#include<stdio.h>
void butler(void);//ANSI/ISO C函数原型
int main(void)
{
printf("I will summon the butler function.\n");
butler();
printf("Yes. Bring me some tea and writeable DVDs.\n");
return 0;
}
void butler(void)//函数定义开始
{
printf("You rang, sir?\n");
}
```

该程序输出如下：

I will summon the butler function.

You rang, sir?

Yes. Bring me some tea and writeable DVDs.

butler() 函数在程序中出现了 3 次。第 1 次是函数原型（prototype），告知编译器在程序中要使用该函数；第二次以函数调用（function call）的形式出现在 main() 中；最后一次出现在函数定义（function definition）中，函数定义既是函数本身的源代码。下面逐一分析。

C90 标准新增了函数原型，旧式的编译器可能无法识别（稍后将介绍），如果使用这种编译器应该怎么做）。函数原型是一种声明形式，告知编译器正在使用某函数，因此函数原型也被称为函数声明（function declaration）。函数原型还指明了函数的属性。例如，butler() 函数原型中的第 1 个 void 表明，butler() 函数没有返回值（通常，被调函数会向主调函数返回一个值），但是 butler() 函数没有）。第 2 个 void(butler(void) 中的 void) 的意思是 butler() 函数不带参数。因此，当编译器运行至此，会检查 butler() 是否使用得当。注意，void 在这里的意思是“空的”，而不是“无效”。

早期的 C 语言支持一种更简单的函数声明，只需指定返回类型，不用描述参数：

```
void butler();
```

早期的 C 代码中的函数声明就类似上面这样，还是现在的函数原型。

C90、C99 和 C11 标准都承认旧版本的形式，但是也表明了会逐渐淘汰这种过时的写法。如果使用以前写的 C 代码，就需要把旧式声明转换成函数原型。后面章节会继续介绍函数原型的相关内容。

接下来继续分析程序。在 main() 中调用 butler() 很简单，写出函数名和圆括号即可。当 butler() 执行完毕后，程序会继续执行 main() 中的下一条语句。

程序的最后部分是 butler() 函数的定义，其形式和 main() 相同，都包含函数头和用花括号括起来的函数体。函数头重述了函数原型的信息：butler() 不带任何参数，且没有返回值。如果使用老式的编译器，请去掉圆括号中的 void。

注意，何时执行 butler() 函数取决于它在 main() 中被调用的位置，而不是 butler() 定义在文件中的位置。例如，把 butler() 函数的定义放在 main() 定义之前，不会改变程序的执行顺序，butler() 函数仍然在两次 printf() 调用之间被调用。记住，无论 main() 在程序文件处于什么位置，所有的 C 程

序都从 main() 开始执行。但是，C 的惯例是把 main() 放在开头，因为它提供了程序的基本框架。

C 标准建议，要为程序中用到的所有函数提供函数原型。标准 include 文件（包含文件）为标准库函数提供可函数原型。例如，在 C 标准中，stdio.h 文件包含了 printf() 的函数原型。第 6 章最后一个示例演示了如何使用带返回值的函数，第 9 章交详细全面地介绍函数。

3.7 2.7 调试程序

现在，可以编写一个简单的 C 程序，但是可能会犯一些简单的错误。程序的错误通常叫做 bug，找出并修正错误的过程叫做调试（debug）。程序清单 2.4 是一个有错误的程序，看看你能找出几处。程序清单 2.4 nogood.c 程序

```
//nogood.c ——有错误的程序
#include<stdio.h>
int main(void)
(
int n,int n2, int n3;//该程序有多处错误
n=5;
n2=n*n;
n3=n2*n2;
printf("n=%d,n squared=%d, n cubed=%d\n",n,n2,n3)
return 0;
)
```

3.7.1 2.7.1 语法错误

程序清单 2.4 中有多处语法错误。如果不遵循 C 语言的规则就会犯语法错误。这类似于英文中的语法错误。例如，看看这个句子：Bugs frustrate be can. 该句子中的英文单词都是有效的单词（即，拼写正确），但是并未按照正确的顺序组织句子，而且用词也不妥。C 语言的语法错误指的是，把有效的 C 符号放在错误的地方。

nogood.c 程序中有哪些错误？其一，main() 函数体使用圆括号来代替花括号。这就是把 C 符号用错了地方。其二，变量声明应该这样写：

int n,n2,n3;

或者，这样写：

int n;

int n2;

int n3;

其三，main() 中的注释末尾漏掉了 */(另一种修改方案是，用//替换，吾即用此)。最后，printf() 语句末尾漏掉了分号。

如何发现程序的语法错误？首先，在编译之前，浏览源代码看是否能发现一些明显的错误。接下来，查看编译器是否发现错误，检查程序的语法错误是它的工作之一。在编译程序时，编译器发现错误会报告错误信息，指出每一处错误的性质和具体位置。

尽管如此，编译器也有出错的时候。也许某处隐藏的语法错误会导致编译器误判。例如，由于 nogood.c 程序未正确声明 n2 和 n3，会导致编译器在使用这些变量时发现更多问题。实际上，有时有用把编译器报告的所有错误逐一修正，仅修正第 1 条或前几处错误后，错误信息就会少很多。继续这样做，直到编译器不再报错。编译器另一个常见的毛病是，报错的位置比真正的错误位置滞后一行。例如，编译器在编译下一行时才会发现上一行缺少分号。因此，如果编译器报错某行缺少分号，请检查上一行。

3.7.2 2.7.2 语义错误

语义错误是指意思上的错误。例如，考虑这个句子：Scornful derivatives sing greenly(轻蔑的衍生物不熟练地唱歌)。句子中的形容词、名词、动词和副词都在正确的位置上，所以语法正确。但是却让人不知所云。在 C 语言中，如果遵循了 C 规则，但是结果不正确，那就是犯了语义错误。程序示例中有这样的错误：

n3=n2*n2;

此处，n3 原意表示 n 的 3 次方，但是代码中的 n3 被设置成 n 的 4 次方（n2=n*n）。

编译器无法检测语义错误，因为这类错误并未违反 C 语言的规则。编译器无法了解你的真正意图，所以你只能自己找出这些错误。例如，假设你修正了程序的语法错误，程序应该如程序清单 2.5 所示：

程序清单 2.5stillbad.c 程序

```
//stillbad.c ——修复了语法错误的程序
#include<stdio.h>
int main(void)
{
int n,n2,n3;
//该训育有一个语义错误
n=5;
n2=n*n;
n3=n2*n2;
printf("n=%d,n squared=%d, n cubed=%d\n",n,n2,n3);
return 0;
}
```

该程序的输出如下：

n=5,n squared=25,n cubed=625

如果对简单的立方比较熟悉，就会注意到 625 不对。下一步是跟踪程序的执行步骤，找出程序如何得出这个答案。对于本例，通过查看代码就会发现其中的错误，但是，还应该学习更系统的方法。方法之一是，把自己想象成计算机，跟着程序的步骤一步一步地执行。下面，来试试这种方法。

main() 函数体一开始就声明了 3 个变量：n,n2,n3. 你可以画出 3 个盒子并把变量名写在盒子上来模拟这种情况（见图 2.6）。接下来，程序把 5 赋给变量 n。何以在标签为 n 的盒子里写上 5. 接着，程序把 n 和 n 相乘，并把乘积赋给 n2。因此，查看标签为 n 的盒子，其值是 5，5 乘以 5 得 25，于是把 25 放进标签为 n2 的盒子里。为了模拟下一条语句（n3=n2*n2）. 查看 n22 盒子，发现其值是 25。25 乘以 25 得 625，把 625 放进标签为 n3 的盒子。原来如此！程序中计算的是 n2 的平方，不是 n2 乘以 n 得到 n 的 3 次方。

对于上面程序示例，检查程序的过程可能过于繁琐。但是，用这种方法一步一步查看程序的执行情况，通常是发现程序问题所在的良方。

***2.7.3 程序状态

通过逐步跟踪程序的执行步骤，并记录每个变量，便可监视程序的状态。程序状态（program state）是在程序的执行过程中，某给定点上所有变量值的集合。它是计算机头前状态的一个快照。

刚刚讨论了一种跟踪程序状态的方法：自己模拟计算机逐步执行程序。但是，如果程序中有 10000 次循环，这种方法恐怕行不通。不过，可以跟踪一小部分循环，看看程序是否按照预期的方式执行。另外，还要考虑一种情况：很可能按照自己所想去执行程序，而不是根据实际写出来的代码去执行。因此，要尽量忠实代码来模拟。

定位主义错误的另一种方法是：在程序中的关键点插入窗外的 printf() 语句，以监视制定变量值的变化。通过查看值的变化可以了解程序的执行情况。对程序的执行满意后，便可删除额外的 printf() 语句，然后重新编译。

检测程序状态的第 3 种方法是使用调试器。调试器（debugger）是一种程序，让你一步一步运行另一个程序，并检查该唾弃变量的值。调试器有不同的使用难度和复杂度。较高级的调试器会显示正在执行的源代码行号。这在检查有多条执行路径的程序时很方便，因为很容易知道正在执行哪条路径。如果你的编译器自带调试器，现在可以花点时间学会怎么使用它。例如，试着调试一下程序清单 2.4.

3.8 2.8 关键字和保留标识符

关键字是 C 语言的词汇。它们以 C 而言比较特殊，不能用它们作为标识符（如，变量名）。许多关键字用于指定不同的类型，如 int。还有一些关键字（如，if ）用于控制程序中语句的执行顺序。在表 2.2 中所列的 C 语言关键字中，粗体表示是 C90 标准新增的关键字，任何表示的 C99 标准新增的关键字，粗斜体表示的是 C11 标准新增的关键字。表 2.2 ISO C 关键字

auto	extern	short	while
break	float	**signed**	***_Alignas***
case	fort	sizeof	***_Alignof***
char	goto	static	***_Atomic***
const	if	struct	***_Bool***
continue	inline	switch	***_Complex***
default	int	typedef	***_Generic***
do	long	union	***_Imaginary***
double	register	unsigned	***_Noreturn***
else	restrict	void	$\textbf{\textit{\_\_Static}}_{\textbf{\textit{assert}}}$
enum	return	*volatile	$\textbf{\textit{\_\_Thread}}_{\textbf{\textit{local}}}$

如果使用关键字不当（如，用关键字作为变量名），编译器会将其视为语法错误。还有一些保留标识符（reserved identifier），C 语言已经指定了它们的用途或保留它们的使用权，如果你使用这些标识符来表示其他意思会导致一些问题。因此，尽管它们也是有效的名称，不会引起语法错误，也不能随便使用。保留标识符包括那些以下划线字符开头的标识符和标准库函数名，如 printf()。