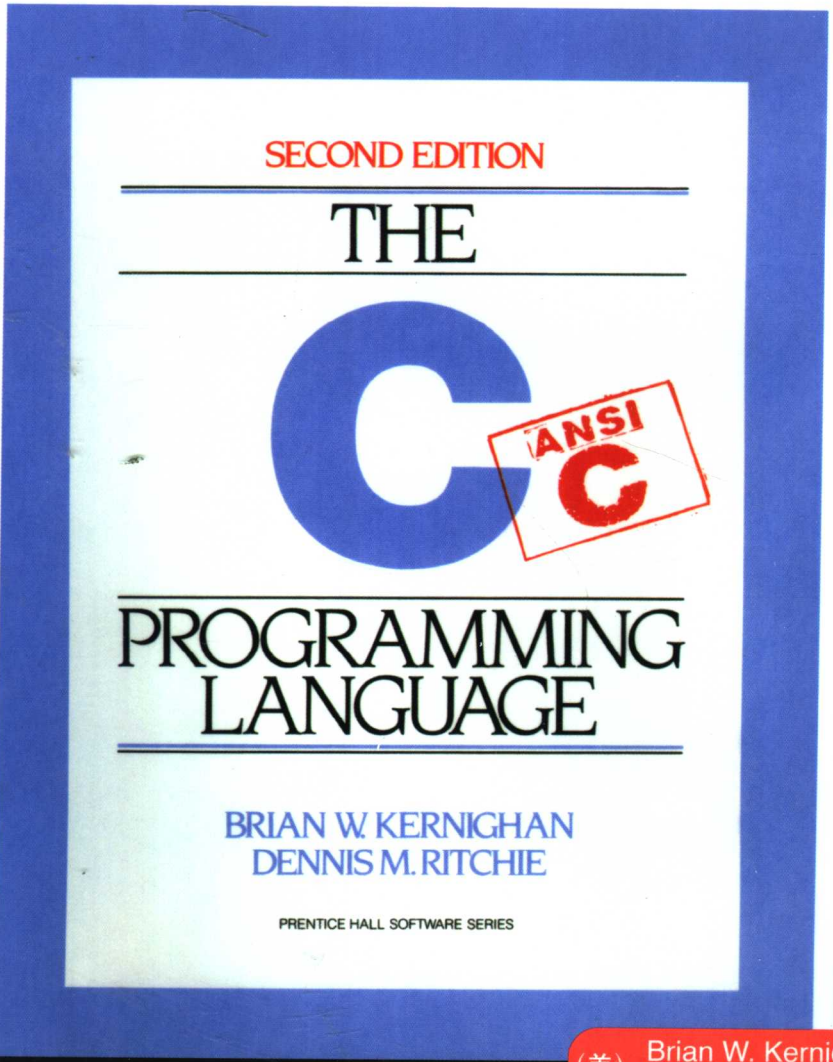
经 典 原 版 书 库

C程序设计语言

（英文版 · 第2版）

K&R C Bible



(美) Brian W. Kernighan
Dennis M. Ritchie 著

机械工业出版社
China Machine Press

经 典 原 版 书 库

C程序设计语言

（英文版 · 第2版）

The C Programming Language

(Second Edition)

江苏工业学院图书馆
藏书章

(美) Brian W. Kernighan
Dennis M. Ritchie 著

机械工业出版社
China Machine Press

Original edition, entitled THE C PROGRAMMING LANGUAGE, 2nd Edition, 0131103628 by KERNIGHAN, BRIAN W.; RITCHIE, DENNIS M., published by Pearson Education, Inc, publishing as Prentice Hall PTR, Copyright © 1988, 1978 by Bell Telephone Laboratories, Incorporated.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

China edition published by PEARSON EDUCATION ASIA LTD., and CHINA MACHINE PRESS, Copyright © 2006.

This edition is manufactured in the People's Republic of China, and is authorized for sale only in People's Republic of China excluding Hong Kong, Macau and Taiwan.

本书英文影印版由Pearson Education Asia Ltd.授权机械工业出版社独家出版。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。

仅限于中华人民共和国境内（不包括中国香港、澳门特别行政区和中国台湾地区）销售发行。

本书封面贴有Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

版权所有，侵权必究。

本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2006-3993

图书在版编目（CIP）数据

C程序设计语言（英文版·第2版）/（美）克尼汉（Kernighan, B. W.）等著. -北京：机械工业出版社，2006.8

（经典原版书库）

书名原文：The C Programming Language, Second Edition

ISBN 7-111-19626-0

Ⅰ. C…　Ⅱ. 克…　Ⅲ. C语言-程序设计-英文　Ⅳ. TP312

中国版本图书馆CIP数据核字（2006）第080926号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：　迟振春

北京瑞德印刷有限公司印刷·新华书店北京发行所发行

2006年8月第1版第1次印刷

170mm × 242mm　·　17.75印张

定价：35.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科

技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzjsj@hzbook.com
联系电话：（010）68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

试读结束：需要全本请在线购买： www.ertongbook.com

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

C 序 C PROGRAMMING LANGUAGE

自从1978年《The C Programming Language》一书出版以来，计算机领域经历了一场革命。大型计算机的功能越来越强大，而个人计算机的性能也可以与十多年前的大型机相媲美。在此期间，C语言也在悄悄地演进，其发展早已超出了它仅仅作为UNIX操作系统的编程语言的初衷。

C语言普及程度的逐渐增加以及该语言本身的发展，加之很多组织开发出了与其设计有所不同的编译器，所有这一切都要求对C语言有一个比本书第1版更精确、更适应其发展的定义。1983年，美国国家标准协会（ANSI）成立了一个委员会，其目标是制定“一个无歧义性的且与具体机器无关的C语言定义”，而同时又要保持C语言原有的“精神”。结果产生了C语言的ANSI标准。

ANSI标准规范了一些在本书第1版中提及但没有具体描述的结构，特别是结构赋值和枚举。该标准还提供了一种新的函数声明形式，允许在使用过程中对函数的定义进行交叉检查。标准中还详细说明了一个具有标准输入/输出、内存管理和字符串操作等扩展函数集的标准库。它精确地说明了在C语言原始定义中并不明晰的某些特性的行为，同时还明确了C语言中与具体机器相关的一些特性。

本书第2版介绍的是ANSI标准定义的C语言。尽管我们已经注意到了该语言中已经变化了的地方，但我们还是决定在这里只列出它们的新形式。最重要的原因是，新旧形式之间并没有太大的差别；最明显的变化是函数的声明和定义。目前的编译器已经能够支持该标准的大部分特性。

我们将尽力保持本书第1版的简洁性。C语言并不是一种大型语言，也不需要用一本很厚的书来描述。我们在讲解一些关键特性（比如指针）时做了改进，它是C语言程序设计的核心。我们重新对以前的例子进行了精炼，并在某些章节中增加了一些新例子。例如，我们通过实例程序对复杂的声明进行处理，以将复杂的声明转换为描述性的说明或反之。像前一版中的例子一样，本版中所有例子都以可被机器读取的文本形式直接通过了测试。

附录A只是一个参考手册，而非标准，我们希望通过较少的篇幅概述标准中的要点。该附录的目的是帮助程序员更好地理解语言本身，而不是为编译器的实现者提供一个精确的定义——这正是语言标准所应当扮演的角色。附录B对标准库提供的功能进行了总结，它同样是面向程序员而非编译器实现者的。附录C对ANSI标准相对于以前版本所做的变更进行了小结。

我们在第1版中曾说过：“随着使用经验的增加，使用者会越来越感到得心应手”。

经过十几年的实践，我们仍然这么认为。我们希望这本书能够帮助读者学好并用好C语言。

非常感谢帮助我们完成本书的朋友们。Jon Bentley、Doug Gwyn、Doug McIlroy、Peter Nelson和Rob Pike几乎对本书手稿的每一页都提出了建议。我们非常感谢Al Aho、Dennis Allison、Joe Campbell、G. R. Emlin、Karen Fortgang、Allen Holub、Andrew Hume、Dave Kristol、John Linderman、Dave Prosser、Gene Spafford和Chris Van Wyk等人，他们仔细阅读了本书。我们也收到了来自Bill Cheswick、Mark Kernighan、Andy Koenig、Robin Lake、Tom London、Jim Reeds、Clovis Tondo和Peter Weinberger等人很好的建议。Dave Prosser为我们回答了很多关于ANSI标准的细节问题。我们大量地使用了Bjarne Stroustrup的C++翻译程序进行程序的局部测试。Dave Kristol为我们提供了一个ANSI C编译器以进行最终的测试。Rich Drechsler帮助我们进行了大量的排版工作。

真诚地感谢每个人！

Brian W. Kernighan
Dennis M. Ritchie

第1版序 C PROGRAMMING LANGUAGE

C语言是一种通用的程序设计语言，其特点包括简洁的表达式、流行的控制流和数据结构、丰富的运算符集等。C语言不是一种“很高级”的语言，也不“庞大”，并且不专用于某一个特定的应用领域。但是，C语言的限制少，通用性强，这使得它比一些公认为功能强大的语言使用更方便、效率更高。

C语言最初是由Dennis Ritchie为UNIX操作系统设计的，并在DEC PDP-11计算机上实现。UNIX操作系统、C编译器和几乎所有的UNIX应用程序（包括编写本书时用到的所有软件）都是用C语言编写的。同时，还有一些适用于其他机器的编译器产品，比如IBM System/370、Honeywell 6000和Interdata 8/32等。但是，C语言不受限于任何特定的机器或系统，使用它可以很容易地编写出不经修改就可以运行在所有支持C语言的机器上的程序。

本书的目的是帮助读者学习如何用C语言编写程序。本书的开头有一个指南性的引言，目的是使新用户能尽快地开始学习；随后在不同的章节中介绍了C语言的各种主要特性；本书的附录中还包括一份参考手册。本书并不仅仅只是讲述语言的一些规则，而是采用阅读别人的代码、自己编写代码、修改某些代码等不同的方式来指导读者进行学习。书中的大部分例子都可以直接完整地运行，而不只是孤立的程序段。所有例子的文本都以可被机器读取的文本形式直接通过了测试。除了演示如何有效地使用语言外，我们还尽可能地在适当的时候向读者介绍一些高效的算法、良好的程序设计风格以及正确的设计原则。

本书并不是一本有关程序设计的入门性手册，它要求读者熟悉基本的程序设计概念，如变量、赋值语句、循环和函数等。尽管如此，初级的程序员仍能够阅读本书，并借此学会C语言。当然，知识越丰富，学习起来就越容易。

根据我们的经验，C语言是一种令人愉快的、具有很强表达能力的通用的语言，适合于编写各种程序。它容易学习，并且随着使用经验的增加，使用者会越来越感到得心应手。我们希望本书能帮助读者用好C语言。

来自许多朋友和同事的中肯批评和建议对本书的帮助很大，也使我们在写作本书过程中受益匪浅。在此特别感谢Mike Bianchi、Jim Blue、Stu Feldman、Doug McIlroy、Bill Roome、Bob Rosin和Larry Rosler等人，他们细心地阅读了本书的多次修改版本。我们在这里还要感谢Al Aho、Steve Bourne、Dan Dvorak、Chuck Haley、Debbie Haley、Marion Harris、Rick Holt、Steve Johnson、John Mashey、Bob Mitze、Ralph Muha、Peter Nelson、Elliot Pinson、Bill Plauger、Jerry Spivack、Ken Thompson和Peter Weinberger等人，他们在不同阶段提出了非常有益的意见，此外还要感谢Mike Lesk和Joe Ossanna，他们在排版方面给予了我们很宝贵的帮助。

Brian W. Kernighan
Dennis M. Ritchie

Contents

试读结束：需要全本请在线购买： www.ertongbook.com

Introduction

C is a general-purpose programming language. It has been closely associated with the UNIX system where it was developed, since both the system and most of the programs that run on it are written in C. The language, however, is not tied to any one operating system or machine; and although it has been called a "system programming language" because it is useful for writing compilers and operating systems, it has been used equally well to write major programs in many different domains.

Many of the important ideas of C stem from the language BCPL, developed by Martin Richards. The influence of BCPL on C proceeded indirectly through the language B, which was written by Ken Thompson in 1970 for the first UNIX system on the DEC PDP-7.

BCPL and B are "typeless" languages. By contrast, C provides a variety of data types. The fundamental types are characters, and integers and floating-point numbers of several sizes. In addition, there is a hierarchy of derived data types created with pointers, arrays, structures, and unions. Expressions are formed from operators and operands; any expression, including an assignment or a function call, can be a statement. Pointers provide for machine-independent address arithmetic.

C provides the fundamental control-flow constructions required for well-structured programs: statement grouping, decision making (`if-else`), selecting one of a set of possible cases (`switch`), looping with the termination test at the top (`while`, `for`) or at the bottom (`do`), and early loop exit (`break`).

Functions may return values of basic types, structures, unions, or pointers. Any function may be called recursively. Local variables are typically "automatic," or created anew with each invocation. Function definitions may not be nested but variables may be declared in a block-structured fashion. The functions of a C program may exist in separate source files that are compiled separately. Variables may be internal to a function, external but known only within a single source file, or visible to the entire program.

A preprocessing step performs macro substitution on program text, inclusion of other source files, and conditional compilation.

C is a relatively "low level" language. This characterization is not

pejorative; it simply means that C deals with the same sort of objects that most computers do, namely characters, numbers, and addresses. These may be combined and moved about with the arithmetic and logical operators implemented by real machines.

C provides no operations to deal directly with composite objects such as character strings, sets, lists, or arrays. There are no operations that manipulate an entire array or string, although structures may be copied as a unit. The language does not define any storage allocation facility other than static definition and the stack discipline provided by the local variables of functions; there is no heap or garbage collection. Finally, C itself provides no input/output facilities; there are no READ or WRITE statements, and no built-in file access methods. All of these higher-level mechanisms must be provided by explicitly-called functions. Most C implementations have included a reasonably standard collection of such functions.

Similarly, C offers only straightforward, single-thread control flow: tests, loops, grouping, and subprograms, but not multiprogramming, parallel operations, synchronization, or coroutines.

Although the absence of some of these features may seem like a grave deficiency ("You mean I have to call a function to compare two character strings?"), keeping the language down to modest size has real benefits. Since C is relatively small, it can be described in a small space, and learned quickly. A programmer can reasonably expect to know and understand and indeed regularly use the entire language.

For many years, the definition of C was the reference manual in the first edition of *The C Programming Language*. In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or "ANSI C," was completed late in 1988. Most of the features of the standard are already supported by modern compilers.

The standard is based on the original reference manual. The language is relatively little changed; one of the goals of the standard was to make sure that most existing programs would remain valid, or, failing that, that compilers could produce warnings of new behavior.

For most programmers, the most important change is a new syntax for declaring and defining functions. A function declaration can now include a description of the arguments of the function; the definition syntax changes to match. This extra information makes it much easier for compilers to detect errors caused by mismatched arguments; in our experience, it is a very useful addition to the language.

There are other small-scale language changes. Structure assignment and enumerations, which had been widely available, are now officially part of the language. Floating-point computations may now be done in single precision. The properties of arithmetic, especially for unsigned types, are clarified. The preprocessor is more elaborate. Most of these changes will have only minor

effects on most programmers.

A second significant contribution of the standard is the definition of a library to accompany C. It specifies functions for accessing the operating system (for instance, to read and write files), formatted input and output, memory allocation, string manipulation, and the like. A collection of standard headers provides uniform access to declarations of functions and data types. Programs that use this library to interact with a host system are assured of compatible behavior. Most of the library is closely modeled on the "standard I/O library" of the UNIX system. This library was described in the first edition, and has been widely used on other systems as well. Again, most programmers will not see much change.

Because the data types and control structures provided by C are supported directly by most computers, the run-time library required to implement self-contained programs is tiny. The standard library functions are only called explicitly, so they can be avoided if they are not needed. Most can be written in C, and except for the operating system details they conceal, are themselves portable.

Although C matches the capabilities of many computers, it is independent of any particular machine architecture. With a little care it is easy to write portable programs, that is, programs that can be run without change on a variety of hardware. The standard makes portability issues explicit, and prescribes a set of constants that characterize the machine on which the program is run.

C is not a strongly-typed language, but as it has evolved, its type-checking has been strengthened. The original definition of C frowned on, but permitted, the interchange of pointers and integers; this has long since been eliminated, and the standard now requires the proper declarations and explicit conversions that had already been enforced by good compilers. The new function declarations are another step in this direction. Compilers will warn of most type errors, and there is no automatic conversion of incompatible data types. Nevertheless, C retains the basic philosophy that programmers know what they are doing; it only requires that they state their intentions explicitly.

C, like any other language, has its blemishes. Some of the operators have the wrong precedence; some parts of the syntax could be better. Nonetheless, C has proven to be an extremely effective and expressive language for a wide variety of programming applications.

The book is organized as follows. Chapter 1 is a tutorial on the central part of C. The purpose is to get the reader started as quickly as possible, since we believe strongly that the way to learn a new language is to write programs in it. The tutorial does assume a working knowledge of the basic elements of programming; there is no explanation of computers, of compilation, nor of the meaning of an expression like n=n+1. Although we have tried where possible to show useful programming techniques, the book is not intended to be a reference work on data structures and algorithms; when forced to make a choice, we have concentrated on the language.

Chapters 2 through 6 discuss various aspects of C in more detail, and rather more formally, than does Chapter 1, although the emphasis is still on examples of complete programs, rather than isolated fragments. Chapter 2 deals with the basic data types, operators and expressions. Chapter 3 treats control flow: `if-else`, `switch`, `while`, `for`, etc. Chapter 4 covers functions and program structure—external variables, scope rules, multiple source files, and so on—and also touches on the preprocessor. Chapter 5 discusses pointers and address arithmetic. Chapter 6 covers structures and unions.

Chapter 7 describes the standard library, which provides a common interface to the operating system. This library is defined by the ANSI standard and is meant to be supported on all machines that support C, so programs that use it for input, output, and other operating system access can be moved from one system to another without change.

Chapter 8 describes an interface between C programs and the UNIX operating system, concentrating on input/output, the file system, and storage allocation. Although some of this chapter is specific to UNIX systems, programmers who use other systems should still find useful material here, including some insight into how one version of the standard library is implemented, and suggestions on portability.

Appendix A contains a language reference manual. The official statement of the syntax and semantics of C is the ANSI standard itself. That document, however, is intended foremost for compiler writers. The reference manual here conveys the definition of the language more concisely and without the same legalistic style. Appendix B is a summary of the standard library, again for users rather than implementers. Appendix C is a short summary of changes from the original language. In cases of doubt, however, the standard and one's own compiler remain the final authorities on the language.

CHAPTER 1: A Tutorial Introduction

Let us begin with a quick introduction to C. Our aim is to show the essential elements of the language in real programs, but without getting bogged down in details, rules, and exceptions. At this point, we are not trying to be complete or even precise (save that the examples are meant to be correct). We want to get you as quickly as possible to the point where you can write useful programs, and to do that we have to concentrate on the basics: variables and constants, arithmetic, control flow, functions, and the rudiments of input and output. We are intentionally leaving out of this chapter features of C that are important for writing bigger programs. These include pointers, structures, most of C's rich set of operators, several control-flow statements, and the standard library.

This approach has its drawbacks. Most notable is that the complete story on any particular language feature is not found here, and the tutorial, by being brief, may also be misleading. And because the examples do not use the full power of C, they are not as concise and elegant as they might be. We have tried to minimize these effects, but be warned. Another drawback is that later chapters will necessarily repeat some of this chapter. We hope that the repetition will help you more than it annoys.

In any case, experienced programmers should be able to extrapolate from the material in this chapter to their own programming needs. Beginners should supplement it by writing small, similar programs of their own. Both groups can use it as a framework on which to hang the more detailed descriptions that begin in Chapter 2.

1.1 Getting Started

The only way to learn a new programming language is by writing programs in it. The first program to write is the same for all languages:

Print the words

```
hello, world
```

This is the big hurdle; to leap over it you have to be able to create the program

试读结束：需要全本请在线购买： www.ertongbook.com