

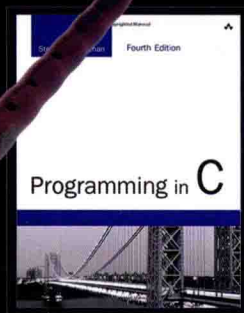
C语言程序设计

(英文版)

(第4版)

Programming in C (4th Edition)

[美] Stephen G. Kochan 著



· 原味精品书系 ·

C语言程序设计

(英文版)

(第4版)

Programming in C (4th Edition)

[美] Stephen G. Kochan 著

电子工业出版社
Publishing House of Electronics Industry
北京·BEIJING

内容简介

本书全面介绍了C语言的各种特性,包括C11中增加的内容。书中包含大量完整的示例及详细的讲解。附录中详尽总结了C语言和C语言库,两者的组织形式都便于快速参考。

本书通过示例来教授C语言,每个新概念都有完整的C程序做演示。你不仅可以学习这门语言的基础知识,还能养成良好的程序设计习惯。另外,每章最后附有习题,便于课堂学习或自学。

无论是否拥有编程经验,你都可以通过本书透彻地理解C语言。

Original edition, entitled Programming in C, 4E, 9780321776419 by Stephen G. Kochan, published by Pearson Education, Inc., publishing as Addison-Wesley, Copyright © 2015 Pearson Education, Inc.

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 Publishing House of Electronics Industry Copyright © 2016. The edition is manufactured in the People's Republic of China, and is authorized for sale and distribution only in the mainland of China exclusively (except Hong Kong SAR, Macau SAR, and Taiwan).

本书英文影印版专有出版权由 Pearson Education 培生教育出版亚洲有限公司授予电子工业出版社。未经出版者预先书面许可,不得以任何方式复制或抄袭本书的任何部分。

本书仅限中国大陆境内(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。

本书英文影印版贴有 Pearson Education 培生教育出版集团激光防伪标签,无标签者不得销售。

版权贸易合同登记号 图字:01-2015-5599

图书在版编目(CIP)数据

C语言程序设计:第4版=Programming in C, 4E:英文/(美)寇肯(Kochan, S.G.)著.—北京:电子工业出版社,2016.4

(原味精品书系)

ISBN 978-7-121-27319-3

I. ① C…II. ① 寇…III. ① C语言—程序设计—英文IV. ① TP312

中国版本图书馆CIP数据核字(2015)第234566号

策划编辑:张春雨 刘芸

责任编辑:徐津平

印刷:三河市华成印务有限公司

装订:三河市华成印务有限公司

出版发行:电子工业出版社

北京市海淀区万寿路173信箱 邮编:100036

开本:787×980 1/16 印张:33.75 字数:650千字

版次:2016年4月第1版

印次:2016年4月第1次印刷

定价:108.00元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 zltz@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线:(010) 88258888。

献给我的父亲和母亲

关于作者

Stephen G. Kochan 使用 C 语言设计软件已经有超过 30 年的时间。他是几本最畅销的 C 语言书籍的作者，包括 *Programming in C*、*Programming in Objective-C* 和 *Topics in C Programming*。他在 UNIX 方面也有大量著述，是 *Exploring the UNIX System* 和 *UNIX Shell Programming* 等书的著者或合著者。

对本书有贡献的作者

Dean Miller 是一位作家和编辑，在出版和特许消费者产品领域都拥有超过 20 年的经验。他是 *Sams Teach Yourself C in One Hour a Day* 和 *Sams Teach Yourself Beginning Programming in 24 Hours* 最近几个版本的合著者。

致谢

衷心地感谢下列各位在我准备本书各个版本时提供的帮助：Douglas McCormick、Jim Scharf、Henry Tabickman、Dick Fritz、Steve Levy、Tony Ianinno 和 Ken Brown。还要感谢纽约大学的 Henry Mullish 在写作方面给我的诸多教益，感谢他带我走进出版领域。

在 Pearson 的工作人员中，我要特别感谢 Mark Taber 和责任编辑 Mandie Frank，还要感谢文字编辑 Charlotte Kughen 和技术编辑 Siddhartha Singh。最后，在 Pearson 还有许多参与本项目但没有与我直接接触的工作人员，在此向他们所有人表示感谢。

目录

Introduction 1

1 Some Fundamentals 5

- Programming 5
- Higher-Level Languages 5
- Operating Systems 6
- Compiling Programs 7
- Integrated Development Environments 10
- Language Interpreters 10

2 Compiling and Running Your First Program 11

- Compiling Your Program 12
- Running Your Program 12
- Understanding Your First Program 13
- Displaying the Values of Variables 15
- Comments 17
- Exercises 19

3 Variables, Data Types, and Arithmetic Expressions 21

- Understanding Data Types and Constants 21
 - The Integer Type `int` 22
 - The Floating Number Type `float` 23
 - The Extended Precision Type `double` 23
 - The Single Character Type `char` 24
 - The Boolean Data Type `_Bool` 24
 - Type Specifiers: `long`, `long long`, `short`, `unsigned`, and `signed` 26
- Working with Variables 29
- Working with Arithmetic Expressions 30
 - Integer Arithmetic and the Unary Minus Operator 33
- Combining Operations with Assignment: The Assignment Operators 39
- Types `_Complex` and `_Imaginary` 40
- Exercises 40

4 Program Looping 43

- Triangular Numbers 43
- The `for` Statement 44
 - Relational Operators 46
 - Aligning Output 50
- Program Input 51
 - Nested `for` Loops 53
 - `for` Loop Variants 55
- The `while` Statement 56
- The `do` Statement 60
 - The `break` Statement 62
 - The `continue` Statement 62
- Exercises 63

5 Making Decisions 65

- The `if` Statement 65
 - The `if-else` Construct 69
 - Compound Relational Tests 72
 - Nested `if` Statements 74
 - The `else if` Construct 76
- The `switch` Statement 83
- Boolean Variables 86
- The Conditional Operator 90
- Exercises 92

6 Working with Arrays 95

- Defining an Array 96
 - Using Array Elements as Counters 100
 - Generating Fibonacci Numbers 103
 - Using an Array to Generate Prime Numbers 104
- Initializing Arrays 106
- Character Arrays 108
 - Base Conversion Using Arrays 109
 - The `const` Qualifier 111
- Multidimensional Arrays 113
- Variable Length Arrays 115
- Exercises 117

7 Working with Functions 119

Defining a Function 119

Arguments and Local Variables 123

 Function Prototype Declaration 124

 Automatic Local Variables 124

Returning Function Results 126

Functions Calling Functions Calling... 130

 Declaring Return Types and Argument Types 133

 Checking Function Arguments 135

Top-Down Programming 137

Functions and Arrays 137

 Assignment Operators 141

 Sorting Arrays 143

 Multidimensional Arrays 146

Global Variables 151

Automatic and Static Variables 155

Recursive Functions 158

Exercises 161

8 Working with Structures 163

The Basics of Structures 163

A Structure for Storing the Date 164

 Using Structures in Expressions 166

Functions and Structures 169

 A Structure for Storing the Time 175

Initializing Structures 178

 Compound Literals 178

Arrays of Structures 180

Structures Containing Structures 183

Structures Containing Arrays 185

Structure Variants 189

Exercises 190

9 Character Strings 193

Revisiting the Basics of Strings 193

Arrays of Characters 194

Variable-Length Character Strings	197
Initializing and Displaying Character Strings	199
Testing Two Character Strings for Equality	202
Inputting Character Strings	204
Single-Character Input	206
The Null String	211
Escape Characters	215
More on Constant Strings	217
Character Strings, Structures, and Arrays	218
A Better Search Method	221
Character Operations	226
Exercises	229
10 Pointers	233
Pointers and Indirection	233
Defining a Pointer Variable	234
Using Pointers in Expressions	237
Working with Pointers and Structures	239
Structures Containing Pointers	241
Linked Lists	243
The Keyword <code>const</code> and Pointers	251
Pointers and Functions	252
Pointers and Arrays	258
A Slight Digression About Program Optimization	262
Is It an Array or Is It a Pointer?	262
Pointers to Character Strings	264
Constant Character Strings and Pointers	266
The Increment and Decrement Operators Revisited	267
Operations on Pointers	271
Pointers to Functions	272
Pointers and Memory Addresses	273
Exercises	275
11 Operations on Bits	277
The Basics of Bits	277
Bit Operators	278
The Bitwise AND Operator	279

- The Bitwise Inclusive-OR Operator 281
- The Bitwise Exclusive-OR Operator 282
- The Ones Complement Operator 283
- The Left Shift Operator 285
- The Right Shift Operator 286
- A Shift Function 286
- Rotating Bits 288
- Bit Fields 291
- Exercises 295

- 12 The Preprocessor 297**
 - The `#define` Statement 297
 - Program Extendability 301
 - Program Portability 302
 - More Advanced Types of Definitions 304
 - The `#` Operator 309
 - The `##` Operator 310
 - The `#include` Statement 311
 - System Include Files 313
 - Conditional Compilation 314
 - The `#ifdef`, `#endif`, `#else`, and `#ifndef` Statements 314
 - The `#if` and `#elif` Preprocessor Statements 316
 - The `#undef` Statement 317
 - Exercises 318

- 13 Extending Data Types with the Enumerated Data Type, Type Definitions, and Data Type Conversions 319**
 - Enumerated Data Types 319
 - The `typedef` Statement 323
 - Data Type Conversions 325
 - Sign Extension 327
 - Argument Conversion 328
 - Exercises 329

- 14 Working with Larger Programs 331**
 - Dividing Your Program into Multiple Files 331
 - Compiling Multiple Source Files from the Command Line 332

Communication Between Modules	334
External Variables	334
Static Versus Extern Variables and Functions	337
Using Header Files Effectively	339
Other Utilities for Working with Larger Programs	341
The <code>make</code> Utility	341
The <code>cv</code> s Utility	343
Unix Utilities: <code>ar</code> , <code>grep</code> , <code>sed</code> , and so on	343
15 Input and Output Operations in C	345
Character I/O: <code>getchar()</code> and <code>putchar()</code>	346
Formatted I/O: <code>printf()</code> and <code>scanf()</code>	346
The <code>printf()</code> Function	346
The <code>scanf()</code> Function	353
Input and Output Operations with Files	358
Redirecting I/O to a File	358
End of File	361
Special Functions for Working with Files	362
The <code>fopen</code> Function	362
The <code>getc()</code> and <code>putc()</code> Functions	364
The <code>fclose()</code> Function	365
The <code>feof</code> Function	367
The <code>fprintf()</code> and <code>fscanf()</code> Functions	367
The <code>fgets()</code> and <code>fputs()</code> Functions	367
<code>stdin</code> , <code>stdout</code> , and <code>stderr</code>	368
The <code>exit()</code> Function	369
Renaming and Removing Files	370
Exercises	371
16 Miscellaneous and Advanced Features	373
Miscellaneous Language Statements	373
The <code>goto</code> Statement	373
The <code>null</code> Statement	374
Working with Unions	375
The Comma Operator	378
Type Qualifiers	379
The <code>register</code> Qualifier	379

- The `volatile` Qualifier 379
- The `restrict` Qualifier 379
- Command-line Arguments 380
- Dynamic Memory Allocation 384
 - The `calloc()` and `malloc()` Functions 385
 - The `sizeof` Operator 385
 - The `free` Function 387
- Exercises 389

- 17 Debugging Programs 391**
 - Debugging with the Preprocessor 391
 - Debugging Programs with `gdb` 397
 - Working with Variables 400
 - Source File Display 401
 - Controlling Program Execution 402
 - Getting a Stack Trace 406
 - Calling Functions and Setting Arrays and Structures 407
 - Getting Help with `gdb` Commands 408
 - Odds and Ends 410

- 18 Object-Oriented Programming 413**
 - What Is an Object Anyway? 413
 - Instances and Methods 414
 - Writing a C Program to Work with Fractions 416
 - Defining an Objective-C Class to Work with Fractions 417
 - Defining a C++ Class to Work with Fractions 421
 - Defining a C# Class to Work with Fractions 424

- A C Language Summary 427**
 - 1.0 Digraphs and Identifiers 427
 - 2.0 Comments 429
 - 3.0 Constants 429
 - 4.0 Data Types and Declarations 432
 - 5.0 Expressions 442
 - 6.0 Storage Classes and Scope 456
 - 7.0 Functions 458
 - 8.0 Statements 460
 - 9.0 The Preprocessor 464

- B The Standard C Library 471**
 - Standard Header Files 471
 - String Functions 474
 - Memory Functions 475
 - Character Functions 476
 - I/O Functions 477
 - In-Memory Format Conversion Functions 482
 - String-to-Number Conversion 483
 - Dynamic Memory Allocation Functions 484
 - Math Functions 485
 - General Utility Functions 493

- C Compiling Programs with gcc 495**
 - General Command Format 495
 - Command-Line Options 496

- D Common Programming Mistakes 499**

- E Resources 505**
 - The C Programming Language 505
 - C Compilers and Integrated Development Environments 506
 - Miscellaneous 507

- Index 509**

Introduction

The C programming language was pioneered by Dennis Ritchie at AT&T Bell Laboratories in the early 1970s. It was not until the late 1970s, however, that this programming language began to gain widespread popularity and support. This was because until that time C compilers were not readily available for commercial use outside of Bell Laboratories. Initially, C's growth in popularity was also spurred on in part by the equal, if not faster, growth in popularity of the Unix operating system. This operating system, which was also developed at Bell Laboratories, had C as its "standard" programming language. In fact, well over 90% of the operating system itself was written in the C language!

The enormous success of the IBM PC and its look-alikes soon made MS-DOS the most popular environment for the C language. As C grew in popularity across different operating systems, more and more vendors hopped on the bandwagon and started marketing their own C compilers. For the most part, their version of the C language was based on an appendix found in the first C programming text—*The C Programming Language*—by Brian Kernighan and Dennis Ritchie. Unfortunately, this appendix did not provide a complete and unambiguous definition of C, meaning that vendors were left to interpret some aspects of the language on their own.

In the early 1980s, a need was seen to standardize the definition of the C language. The American National Standards Institute (ANSI) is the organization that handles such things, so in 1983 an ANSI C committee (called X3J11) was formed to standardize C. In 1989, the committee's work was ratified, and in 1990, the first official ANSI standard definition of C was published.

Because C is used around the world, the International Standard Organization (ISO) soon got involved. They adopted the standard, where it was called ISO/IEC 9899:1990. Since that time, additional changes have been made to the C language. The most recent standard was adopted in 2011. It is known as ANSI C11, or ISO/IEC 9899:2011. It is this version of the language upon which this book is based.

C is a "higher-level language," yet it provides capabilities that enable the user to "get in close" with the hardware and deal with the computer on a much lower level. This is because, although C is a general-purpose structured programming language, it was originally designed with systems programming applications in mind and, as such, provides the user with an enormous amount of power and flexibility.

This book proposes to teach you how to program in C. It assumes no previous exposure to the language and was designed to appeal to novice and experienced programmers alike. If you have previous programming experience, you will find that C has a unique way of doing things that probably differs from other languages you have used.

Every feature of the C language is treated in this text. As each new feature is presented, a small *complete* program example is usually provided to illustrate the feature. This reflects the overriding philosophy that has been used in writing this book: to teach by example. Just as a picture is worth a thousand words, so is a properly chosen program example. If you have access to a computer that supports the C programming language, you are strongly encouraged to download and run each program presented in this book and to compare the results obtained on your system to those shown in the text. By doing so, not only will you learn the language and its syntax, but you will also become familiar with the process of typing in, compiling, and running C programs.

You will find that program readability has been stressed throughout the book. This is because I strongly believe that programs should be written so that they can be easily read—either by the author or by somebody else. Through experience and common sense, you will find that such programs are almost always easier to write, debug, and modify. Furthermore, developing programs that are readable is a natural result of a true adherence to a structured programming discipline.

Because this book was written as a tutorial, the material covered in each chapter is based on previously presented material. Therefore, maximum benefit will be derived from this book by reading each chapter in succession, and you are highly discouraged from “skipping around.” You should also work through the exercises that are presented at the end of each chapter before proceeding on to the next chapter.

Chapter 1, “Some Fundamentals,” which covers some fundamental terminology about higher-level programming languages and the process of compiling programs, has been included to ensure that you understand the language used throughout the remainder of the text. From Chapter 2, “Compiling and Running Your First Program,” on, you will be slowly introduced to the C language. By the time Chapter 15, “Input and Output Operations in C,” rolls around, all the essential features of the language will have been covered. Chapter 15 goes into more depth about I/O operations in C. Chapter 16, “Miscellaneous and Advanced Features,” includes those features of the language that are of a more advanced or esoteric nature.

Chapter 17, “Debugging Programs,” shows how you can use the C preprocessor to help debug your programs. It also introduces you to interactive debugging. The popular debugger `gdb` was chosen to illustrate this debugging technique.

Over the last decade, the programming world has been abuzz with the notion of object-oriented programming, or OOP for short. C is not an OOP language; however, several other programming languages that are based on C are OOP languages. Chapter 18, “Object-oriented Programming,” gives a brief introduction to OOP and some of its terminology. It also gives a brief overview of three OOP languages that are based on C, namely C++, C#, and Objective-C.

Appendix A, “C Language Summary,” provides a complete summary of the language and is provided for reference purposes.

Appendix B, “The Standard C Library,” provides a summary of many of the standard library routines that you will find on all systems that support C.

Appendix C, “Compiling Programs with `gcc`,” summarizes many of the commonly used options when compiling programs with GNU’s C compiler `gcc`.

In Appendix D, “Common Programming Mistakes,” you’ll find a list of common programming mistakes.

Finally, Appendix E, “Resources,” provides a list of resources you can turn to for more information about the C language and to further your studies.

This book makes no assumptions about a particular computer system or operating system on which the C language is implemented. The text makes brief mention of how to compile and execute programs using the popular GNU C compiler `gcc`.