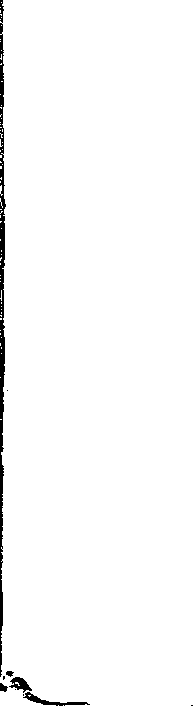
**THE**

PROGRAMMING
  
LANGUAGE



THE

PROGRAMMING
  
LANGUAGE

Brian W. Kernighan
  
Dennis M. Ritchie

*Bell Laboratories*

*Murray Hill, New Jersey*

PRENTICE-HALL, INC., Englewood Cliffs, New Jersey 07632

*Library of Congress Cataloging in Publication Data*

KERNIGHAN, BRIAN W.

The C programming language.

Includes index.

1. C (Computer program language) I. RITCHIE,

DENNIS M., joint author. 11. Title.

QA76.73.C15K47 001.6 424 77-28983
  
ISBN 0-13-110163-3

**The Publisher offers discounts on this book when ordered in bulk** 
  
**quantities. For more information write:**

**Special Sales/College Marketing
  
Prentice-Hall, Inc.**

**College Technical and Reference Division**

**Englewood Cliffs, N.J. 07632**

Copyright @ 1978 by Bell Telephone Laboratories, Incorporated.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval
  
system, or transmitted, in any form or by any means, electronic, mechanical, photocopy­
  
ing, recording, or otherwise, without the prior written permission of the publisher. Print­
  
ed in the United States of America. Published simultaneously in Canada.

This book was set in Times Roman and Courier 12 by the authors, using a Graphic Sys­
  
tems phototypesetter driven by a PDP-11/70 running under the UNIX operating system.

UNIX is a Trademark of Bell Laboratories.

25 24

PRENTICE-HALL INTERNATIONAL, INC., London
  
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, Sydney
  
PRENTICE-HALL OF CANADA, LTD., Toronto

PRENTICE-HALL OF INDIA PRIVATE LIMITED, New Delhi
  
PRENTICE-HALL OF JAPAN, INC., Tokyo

PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., Singapore
  
WHITEHALL BOOKS LIMITED, Wellington, New Zealand

**CONTENTS**

**Preface ix**

**Chapter 0 Introduction 1**

**Chapter 1 A Tutorial Introduction 5**

1.1 Getting Started 5

1.2 Variables and Arithmetic 8

1.3 The For Statement 11

1.4 Symbolic Constants 12

1.5 A Collection of Useful Programs 13

1.6 Arrays 20

1.7 Functions 22

1.8 Arguments — Call by Value 24

1.9 Character Arrays 25

1.10 Scope; External Variables 28

1.11 Summary 31

**Chapter 2 Types, Operators and Expressions 33**

2.1 Variable Names 33

2.2 Data Types and Sizes 33

2.3 Constants 34

2.4 Declarations 36

2.5 Arithmetic Operators 37

2.6 Relational and Logical Operators 38

2.7 Type Conversions 39

2.8 Increment and Decrement Operators 42

2.9 Bitwise Logical Operators **44**

2.10 Assignment Operators and Expressions 46

2.11 Conditional Expressions 47

2.12 Precedence and Order of Evaluation 48

|  |  |  |
| --- | --- | --- |
| Vi THE C PROGRAMMING LANGUAGE | |  |
| **Chapter 3** | **Control Flow** | **51** |
| 3.1 | Statements and Blocks | 51 |
| 3.2 | If-Else | 51 |
| 3.3 | Else-If | 53 |
| 3.4 | Switch | 54 |
| 3.5 | Loops - While and For | 56 |
| 3.6 | Loops - Do-while | 59 |
| 3.7 | Break | 61 |
| 3.8 | Continue | 62 |
| 3.9 | Goto's and Labels | 62 |
| **Chapter 4** | **Functions and Program Structure** | **65** |
| 4.1 | Basics | 65 |
| 4.2 | Functions Returning Non-Integers | 68 |
| 4.3 | More on Function Arguments | 71 |
| 4.4 | External Variables | 72 |
| 4.5 | Scope Rules | 76 |
| 4.6 | Static Variables | 80 |
| 4.7 | Register Variables | 81 |
| 4.8 | Block Structure | 81 |
| 4.9 | Initialization | 82 |
| 4.10 | Recursion | 84 |
| 4.11 | The C Preprocessor | 86 |
| **Chapter 5** | **Pointers and Arrays** | **89** |
| 5.1 | Pointers and Addresses | 89 |
| 5.2 | Pointers and Function Arguments | 91 |
| 5.3 | Pointers and Arrays | 93 |
| 5.4 | Address Arithmetic | 96 |
| 5.5 | Character Pointers and Functions | 99 |
| 5.6 | Pointers are not Integers | 102 |
| 5.7 | Multi-Dimensional Arrays | 103 |
| 5.8 | Pointer Arrays; Pointers to Pointers | 105 |
| 5.9 | Initialization of Pointer Arrays | 109 |
| 5.10 | Pointers vs. Multi-dimensional Arrays | 110 |
| 5.11 | Command-line Arguments | 110 |
| 5.12 | Pointers to Functions | 114 |
| **Chapter 6** | **Structures** | **119** |
| 6.1 | Basics | 119 |
| 6.2 | Structures and Functions | 121 |
| 6.3 | Arrays of Structures | 123 |

|  |  |  |
| --- | --- | --- |
|  | CONTENTS | Vii |
| 6.4 | Pointers to Structures | 128 |
| 6.5 | Self-referential Structures | 130 |
| 6.6 | Table Lookup | 134 |
| 6.7 | Fields | 136 |
| 6.8 | Unions | 138 |
| 6.9 | Typedef | 140 |
| **Chapter 7** | **Input and Output** | **143** |
| 7.1 | Access to the Standard Library | 143 |
| 7.2 | Standard Input and Output - Getchar and Putchar | 144 |
| 7.3 | Formatted Output - Printf | 145 |
| 7.4 | Formatted Input - Scanf | 147 |
| 7.5 | In-memory Format Conversion | 150 |
| 7.6 | File Access | 151 |
| 7.7 | Error Handling - Stderr and Exit | 154 |
| 7.8 | Line Input and Output | 155 |
| 7.9 | Some Miscellaneous Functions | 156 |
| **Chapter 8** | **The UNIX System Interface** | **159** |
| 8.1 | File Descriptors | 159 |
| 8.2 | Low Level I/O - Read and Write | 160 |
| 8.3 | Open, Creat, Close, Unlink | 162 |
| 8.4 | Random Access - Seek and Lseek | 164 |
| 8.5 | Example - An Implementation of Fopen and Getc | 165 |
| 8.6 | Example - Listing Directories | 169 |
| 8.7 | Example - A Storage Allocator | 173 |
| **Appendix A** | **C Reference Manual** | **179** |
| 1. | Introduction | 179 |
| 2. | Lexical conventions | 179 |
| 3. | Syntax notation | 182 |
| 4. | What's in a name? | 182 |
| 5. | Objects and lvalues | 183 |
| 6. | Conversions | 183 |
| 7. | Expressions | 185 |
| 8. | Declarations | 192 |
| 9. | Statements | 201 |
| 10. | External definitions | 204 |
| 11. | Scope rules | 205 |
| 12. | Compiler control lines | 207 |
| 13. | Implicit declarations | 208 |
| 14. | Types revisited | 209 |
| 15. | Constant expressions | 211 |

viii THE C PROGRAMMING LANGUAGE

1. Portability considerations 211
2. Anachronisms 212
3. Syntax Summary 214

**Index 221**

CHAPTER 0: **INTRODUCTION**

C is a general-purpose programming language. It has been closely asso­
  
ciated with the UNIX system, since it was developed on that system, and
  
since UNIX and its software 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 operating
  
systems, it has been used equally well to write major numerical, text-
  
processing, and data-base programs.

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 usual arithmetic and logical opera­
  
tors implemented by actual machines.

C provides no operations to deal directly with composite objects such as
  
character strings, sets, lists, or arrays considered as a whole. There is no
  
analog, for example, of the **PIM** operations which manipulate an entire
  
array or string. 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 like that pro­
  
vided by Algol 68. Finally, C itself provides no input-output facilities: there
  
are no READ or WRITE statements, and no wired-in file access methods.
  
All of these higher-level mechanisms must be provided by explicitly-called
  
functions.

Similarly, C offers only straightforward, single-thread control flow con­
  
structions: tests, loops, grouping, and subprograms, but not multiprogram­
  
ming, 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 dimensions has brought
  
real benefits. Since C is relatively small, it can be described in a small
  
space, and learned quickly. A compiler for C can be simple and compact.
  
Compilers are also easily written; using current technology, one can expect
  
to prepare a compiler for a new machine in a couple of months, and to find

1

2 ' THE C PROGRAMMING LANGUAGE CHAPTER 0

that 80 percent of the code of a new compiler is common with existing ones.
  
This provides a high degree of language mobility. Because the data types
  
and control structures provided by C are supported directly by most existing
  
computers, the run-time library required to implement self-contained pro­
  
grams is tiny. On the PDP-11, for example, it contains only the routines to
  
do 32-bit multiplication and division and to perform the subroutine entry
  
and exit sequences. Of course, each implementation provides a comprehen­
  
sive, compatible library of functions to carry out I/O, string handling, and
  
storage allocation operations, but since they are called only explicitly, they
  
can be avoided if required; they can also be written portably in C itself.

Again because the language reflects the capabilities of current comput­
  
ers, C programs tend to be efficient enough that there is no compulsion to
  
write assembly language instead. The most obvious example of this is the
  
UNIX operating system itself, which is written almost entirely in C. Of
  
13000 lines of system code, only about 800 lines at the very lowest level are
  
in assembler. In addition, essentially all of UNIX applications software is
  
written in C; the vast majority of UNIX users (including one of the authors
  
of this book) do not even know the PDP-11 assembly language.

Although C matches the capabilities of many computers, it is indepen­
  
dent of any particular machine architecture, and so with a little care it is
  
easy to write "portable" programs, that is, programs which can be run
  
without change on a variety of hardware. It is now routine in our environ­
  
ment that software developed on UNIX is transported to the local
  
Honeywell, IBM and Interdata systems. In fact, the C compilers and run­
  
time support on these four machines are much more compatible than the
  
supposedly ANSI standard versions of Fortran. The UNIX operating system
  
itself now runs on both the PDP-11 and the Interdata 8/32. Outside of pro­
  
grams which are necessarily somewhat machine-dependent like the compiler,
  
assembler, and debugger, the software written in C is identical on both
  
machines. Within the operating system itself, the 7000 lines of code outside
  
of the assembly language support and the I/O device handlers is about 95
  
percent identical.

For programmers familiar with other languages, it may prove helpful to
  
mention a few historical, technical, and philosophical aspects of C, for con­
  
trast and comparison.

Many of the most important ideas of C stem from the considerably
  
older, but still quite vital, 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 PDP-7.

Although it shares several characteristic features with **BCPL, C** is in no
  
sense a dialect of it. BCPL and B are "typeless" languages: the only data
  
type is the machine word, and access to other kinds of objects is by special

CHAPTER 0 INTRODUCTION 3

operators or function calls. In C, the fundamental data objects are charac­
  
ters, integers of several sizes, and floating point numbers. In addition, there
  
is a hierarchy of derived data types created with pointers, arrays, structures,
  
unions, and functions.

C provides the fundamental flow-control constructions required for
  
well-structured programs: statement grouping; decision making (if); loop­
  
ing with the termination test at the top **(while, for),** or at the bottom
  
(do); and selecting one of a set of possible cases (switch). (All of these
  
were provided in **BCPL** as well, though with somewhat different syntax; that
  
language anticipated the vogue for "structured programming" by several
  
years.)

C provides pointers and the ability to do address arithmetic. The argu­
  
ments to functions are passed by copying the value of the argument, and it
  
is impossible for the called function to change the actual argument in the
  
caller. When it is desired to achieve "call by reference," a pointer may be
  
passed explicitly, and the function may change the object to which the
  
pointer points. Array names are passed as the location of the array origin,
  
so array arguments are effectively call by reference.

Any function may be called recursively, and its local variables are typi­
  
cally "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 be compiled
  
separately. Variables may be internal to a function, external but known only
  
within a single source file, or completely global. Internal variables may be
  
automatic or static. Automatic variables may be placed in registers for
  
increased efficiency, but the register declaration is only a hint to the com­
  
piler, and does not refer to specific machine registers.

C is not a strongly-typed language in the sense of Pascal or Algol 68. It
  
is relatively permissive about data conversion, although it will not automati­
  
cally convert data types with the wild abandon of **PL/I.** Existing compilers
  
provide no run-time checking of array subscripts, argument types, etc.

For those situations where strong type checking is desirable, a separate
  
version of the compiler is used. This program is called *lint,* apparently
  
because it picks bits of fluff from one's programs. *lint* does not generate
  
code, but instead applies a very strict check to as many aspects of a program
  
as can be verified at compile and load time. It detects type mismatches,
  
inconsistent argument usage, unused or apparently uninitialized variables,
  
potential portability difficulties, and the like. Programs which pass
  
unscathed through *lint* enjoy, with few exceptions, freedom from type errors
  
about as complete as do, for example, Algol 68 programs. We will mention
  
other *lint* capabilities as the occasion arises.

Finally, C, like any other language, has its blemishes. Some of the
  
operators have the wrong precedence; some parts of the syntax could be
  
better; there are several versions of the language extant, differing in minor

4 THE C PROGRAMMING LANGUAGE CHAPTER 0

ways. Nonetheless, C has proven to be an extremely effective and expres­
  
sive language for a wide variety of programming applications.

The rest of the book is organized as follows. Chapter 1 is a tutorial
  
introduction to the central part of C. The purpose is to get the reader
  
started as quickly as possible, since we believe strongly that the only 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 expla­
  
nation of computers, of compilation, nor of the meaning of an expression
  
like n=n+1. Although we have tried where possible to show useful pro­
  
gramming techniques, the book is not intended to be a reference work on
  
data structures and algorithms; when forced to a choice, we have concen­
  
trated 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, useful programs, rather than isolated fragments.
  
Chapter 2 deals with the basic data types, operators and expressions.
  
Chapter 3 treats control flow: if—else, while, for, etc. Chapter 4 cov­
  
ers functions and program structure — external variables, scope rules, and
  
so on. Chapter 5 discusses pointers and address arithmetic. Chapter 6 con­
  
tains the details of structures and unions.

Chapter 7 describes the standard C I/O library, which provides a com­
  
mon interface to the operating system. This I/O library is supported on all
  
machines that support C, so programs which use it for input, output, and
  
other system functions can be moved from one system to another essentially
  
without change.

Chapter 8 describes the interface between C programs and the UNIX
  
operating system, concentrating on input/output, the file system, and porta­
  
bility. Although some of this chapter is UNIX-specific, programmers who
  
are not using a UNIX system should still find useful material here, including
  
some insight into how one version of the standard library is implemented,
  
and suggestions on achieving portable code.

Appendix A contains the C reference manual. This is the "official"
  
statement of the syntax and semantics of C, and (except for one's own com­
  
piler) the final arbiter of any ambiguities and omissions from the earlier
  
chapters.

Since C is an evolving language that exists on a variety of systems, some
  
of the material in this book may not correspond to the current state of
  
development for a particular system. We have tried to steer clear of such
  
problems, and to warn of potential difficulties. When in doubt, however, we
  
have generally chosen to describe the PDP-11 UNIX situation, since that is
  
the environment of the majority of C programmers. Appendix A also
  
describes implementation differences on the major C systems.

**PREFACE**

C is a general-purpose programming language which features economy
  
of expression, modern control flow and data structures, and a rich set of
  
operators. C is not a "very high level" language, nor a "big" one, and is
  
not specialized to any particular area of application. But its absence of res­
  
trictions and its generality make it more convenient and effective for many
  
tasks than supposedly more powerful languages.

C was originally designed for and implemented on the UNIXt operating
  
system on the DEC PDP-11, by Dennis Ritchie. The operating system, the
  
C compiler, and essentially all UNIX applications programs (including all of
  
the software used to prepare this book) are written in C. Production com­
  
pilers also exist for several other machines, including the IBM System/370,
  
the Honeywell 6000, and the Interdata 8/32. C is not tied to any particular
  
hardware or system, however, and it is easy to write programs that will run
  
without change on any machine that supports C.

This book is meant to help the reader learn how to program in C. It
  
contains a tutorial introduction to get new users started as soon as possible,
  
separate chapters on each major feature, and a reference manual. Most of
  
the treatment is based on reading, writing and revising examples, rather
  
than on mere statements of rules. For the most part, the examples are com­
  
plete, real programs, rather than isolated fragments. All examples have
  
been tested directly from the text, which is in machine-readable form.
  
Besides showing how to make effective use of the language, we have also
  
tried where possible to illustrate useful algorithms and principles of good
  
style and sound design.

The book is not an introductory programming manual; it assumes some
  
familiarity with basic programming concepts like variables, assignment state­
  
ments, loops, and functions. Nonetheless, a novice programmer should be
  
able to read along and pick up the language, although access to a more

t UNIX is a Trademark of Bell Laboratories. The UNIX operating system is
  
available under license from Western Electric, Greensboro, N. C.

ix

X THE C PROGRAMMING LANGUAGE

knowledgeable colleague will help.

In our experience, C has proven to be a pleasant, expressive, and versa­
  
tile language for a wide variety of programs. It is easy to learn, and it wears
  
well as one's experience with it grows. We hope that this book will help you
  
to use it well.

The thoughtful criticisms and suggestions of many friends and col­
  
leagues have added greatly to this book and to our pleasure in writing it. In
  
particular, Mike Bianchi, Jim Blue, Stu Feldman, Doug McIlroy, Bill
  
Roome, Bob Rosin, and Larry Rosler all read multiple versions with care.
  
We are also indebted to 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, and Peter Weinberger for helpful comments at various
  
stages, and to Mike Lesk and Joe Ossanna for invaluable assistance with
  
typesetting.

Brian W. Kernighan
  
Dennis M. Ritchie