SECOND EDITION

THE C PROGRAMMING LANGUAGE

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

PRENTICE HALL SOFTWARE SERIES

Preface

The computing world has undergone a revolution since the publication of *The C Programming Language* in 1978. Big computers are much bigger, and personal computers have capabilities that rival mainframes of a decade ago. During this time, C has changed too, although only modestly, and it has spread far beyond its origins as the language of the UNIX operating system.

The growing popularity of C, the changes in the language over the years, and the creation of compilers by groups not involved in its design, combined to demonstrate a need for a more precise and more contemporary definition of the language than the first edition of this book provided. In 1983, the American National Standards Institute (ANSI) established a committee whose goal was to produce ``an unambiguous and machine-independent definition of the language C'', while still retaining its spirit. The result is the ANSI standard for C.

The standard formalizes constructions that were hinted but not described in the first edition, particularly structure assignment and enumerations. It provides a new form of function declaration that permits cross-checking of definition with use. It specifies a standard library, with an extensive set of functions for performing input and output, memory management, string manipulation, and similar tasks. It makes precise the behavior of features that were not spelled out in the original definition, and at the same time states explicitly which aspects of the language remain machine-dependent.

This Second Edition of *The C Programming Language* describes C as defined by the ANSI standard. Although we have noted the places where the language has evolved, we have chosen to write exclusively in the new form. For the most part, this makes no significant difference; the most visible change is the new form of function declaration and definition. Modern compilers already support most features of the standard.

We have tried to retain the brevity of the first edition. C is not a big language, and it is not well served by a big book. We have improved the exposition of critical features, such as pointers, that are central to C programming. We have refined the original examples, and have added new examples in several chapters. For instance, the treatment of complicated declarations is augmented by programs that convert declarations into words and vice versa. As before, all examples have been tested directly from the text, which is in machine-readable form.

Appendix A, the reference manual, is not the standard, but our attempt to convey the essentials of the standard in a smaller space. It is meant for easy comprehension by programmers, but not as a definition for compiler writers -- that role properly belongs to the standard itself. Appendix B is a summary of the facilities of the standard library. It too is meant for reference by programmers, not implementers. Appendix C is a concise summary of the changes from the original version.

As we said in the preface to the first edition, C ``wears well as one's experience with it grows''. With a decade more experience, we still feel that way. We hope that this book will help you learn C and use it well.

We are deeply indebted to friends who helped us to produce this second edition. Jon Bently, Doug Gwyn, Doug McIlroy, Peter Nelson, and Rob Pike gave us perceptive comments on almost every page of draft manuscripts. We are grateful for careful reading by Al Aho, Dennis Allison, Joe Campbell, G.R. Emlin, Karen Fortgang, Allen Holub, Andrew Hume, Dave Kristol, John Linderman, Dave Prosser, Gene Spafford, and Chris van Wyk. We also received helpful suggestions from Bill Cheswick, Mark Kernighan, Andy Koenig, Robin Lake, Tom London, Jim Reeds, Clovis Tondo, and Peter Weinberger. Dave Prosser answered many detailed questions about the ANSI standard. We used Bjarne Stroustrup's C++ translator extensively for local testing of our programs, and Dave Kristol provided us with an ANSI C compiler for final testing. Rich Drechsler helped greatly with typesetting.

Our sincere thanks to all.

Brian W. Kernighan
Dennis M. Ritchie

Preface to the first edition

C is a general-purpose programming language with features economy of expression, modern flow control 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 restrictions 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 UNIX 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 compilers 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 complete, 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 statements, loops, and functions. Nonetheless, a novice programmer should be able to read along and pick up the language, although access to more knowledgeable colleague will help.

In our experience, C has proven to be a pleasant, expressive and versatile language for a wide variety of programs. It is easy to learn, and it wears well as on'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 colleagues 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 volumes 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 Mile Lesk and Joe Ossanna for invaluable assistance with typesetting.

Brian W. Kernighan
Dennis M. Ritchie

Chapter 1 - A Tutorial Introduction

Let us begin with a quick introduction in 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 and its drawbacks. Most notable is that the complete story on any particular 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 a big hurdle; to leap over it you have to be able to create the program text somewhere, compile it successfully, load it, run it, and find out where your output went. With these mechanical details mastered, everything else is comparatively easy.

In C, the program to print ``hello, world'' is

```
#include <stdio.h>

main()
{
  printf("hello, world\n");
}
```

Just how to run this program depends on the system you are using. As a specific example, on the UNIX operating system you must create the program in a file whose name ends in ``.c'', such as `hello.c`, then compile it with the command

```
cc hello.c
```

If you haven't botched anything, such as omitting a character or misspelling something, the compilation will proceed silently, and make an executable file called `a.out`. If you run `a.out` by typing the command

```
a.out
```

it will print

```
hello, world
```

On other systems, the rules will be different; check with a local expert.

Now, for some explanations about the program itself. A C program, whatever its size, consists of *functions* and *variables*. A function contains *statements* that specify the computing operations to be done, and variables store values used during the computation. C functions are like the subroutines and functions in Fortran or the procedures and functions of Pascal. Our example is a function named `main`. Normally you are at liberty to give functions whatever names you like, but ``main" is special - your program begins executing at the beginning of main. This means that every program must have a `main` somewhere.

`main` will usually call other functions to help perform its job, some that you wrote, and others from libraries that are provided for you. The first line of the program,

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

tells the compiler to include information about the standard input/output library; the line appears at the beginning of many C source files. The standard library is described in Chapter 7 and Appendix B.

One method of communicating data between functions is for the calling function to provide a list of values, called *arguments*, to the function it calls. The parentheses after the function name surround the argument list. In this example, `main` is defined to be a function that expects no arguments, which is indicated by the empty list `( )`.

```
#include <stdio.h>                  include information about standard
library
main()                              define a function called main
                                    that received no argument values
{                                   statements of main are enclosed in braces
    printf("hello, world\n");       main calls library function printf
                                    to print this sequence of characters
}                                   \n represents the newline character
```

The first C program

The statements of a function are enclosed in braces `{ }`. The function `main` contains only one statement,

```
printf("hello, world\n");
```