[Back to the Preface to the First Edition](preface1.html) -- [Index](kandr.html) -- [Chapter 1](chapter1.html)

# Introduction

C is a general-purpose programming language. It has been closely associated
with the UNIX operating 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 values (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 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 in late 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 the 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 in 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 ben 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 threats
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 the C language 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.

[Back to the Preface to the First Edition](preface1.html) -- [Index](kandr.html) -- [Chapter 1](chapter1.html)