

# Preface

There are many C books on the market, covering a broad range of applications and platforms, and each comes with some form of cross-reference index. However, the art of indexing a book is quite specialized and, as such, is often done by someone who is not intimately familiar with the book's subject matter. Also, the index of any given book is obviously limited to the subject matter covered in that book. A book on data structures in C, for example, is unlikely to have an index entry for the standard date and time functions.

Many C programmers accumulate more than a few C books during their careers. Of course, none of these books can be all things to all people. Therefore, to research a number of specific terms, the reader may have to scan through the indexes of five or six books to find the relevant information. And in the case of more advanced topics (such as lvalues and sequence points), index entries may be missing altogether. This is where this dictionary becomes useful. Organized in alphabetical order, it allows terms to be found quickly and easily. The format provides for a relatively concise definition, in some cases with a small amount of tutorial material. And, of course, there is extensive cross-referencing to other entries.

## Intended Audience

This book is suitable for C programmers at *all* levels. Because it does not deal with extensions, it is not aimed at any particular kind of application or set of platforms. It is generic and is applicable to anyone working with C using any mainstream implementation of C.

## Limits of This Dictionary

This book is intended as an alphabetized quick reference guide to C. It is not an encyclopedia nor is it intended to replace the C standard or your implementation's documentation. (For information on obtaining a paper or electronic copy of the C standard refer to [www.iso.ch](http://www.iso.ch) or [www.ansi.org](http://www.ansi.org).)

## Acknowledgments

Some thanks are in order: To Dennis Ritchie for creating the C language; to Brian Kernighan for so eloquently writing about C, thereby helping to bootstrap a generation of C programmers; to Jim Brodie, Tom Plum, and P.J. Plauger for so ably steering the original C standards committee, X3J11; to Larry Rosler, David Prosser, Frank Farance, and Larry Jones for their excellent work on editing the C standard; to Randy Hudson and Bill Seymour for editing the Rationale manual; and to all my colleagues on the committees X3J11, NCEG, X3J11.1, J11, and WG14 who have assisted in my numerous C ventures.

Many thanks also go to the reviewers of various drafts of this book: Nelson Beebe, Jim Brodie, Tom MacDonald, Randy Meyers, Tom Plum, Fred Tyde-  
man, and Douglas Walls whose suggestions contributed greatly to the quality  
of this book.

Finally, thanks to Donald Knuth for his typesetting system  $\text{\TeX}$ , and to  
Leslie Lamport for his  $\text{\LaTeX}$  macro package with which this book was pre-  
pared.

## How to Use This Dictionary

Almost every operator and punctuator in C is represented by one or more  
punctuation characters. How, then, can these be assigned an alphabetic entry  
in such a way that they can be found intuitively? I have listed these punc-  
tuation characters in a table that follows this section. Each symbol has a  
corresponding English word or phrase that is the primary entry in the dic-  
tionary for that symbol. Symbols that have multiple meanings have multiple  
entries listed in alphabetical order. The symbols themselves are listed in the  
table in ASCII order. While this in itself is not particularly helpful, neither is  
any other collating sequence.

Many topics have multiple entries because they have more than one obvious  
spelling. For example, `case` labels are listed under both “`case` label” and  
“label, `case`.” Hopefully, in most cases, the primary entry will be the most  
obvious. However, there are numerous entries about `case` and also about  
labels, so some (perhaps seemingly) arbitrary choices had to be made in this  
regard.

Consider the following entry:

---

**addition assignment operator** A binary operator, `+=`, that permits addi-  
tion and assignment to be combined such that *exp1 += exp2* is equivalent  
to *exp1 = exp1 + exp2* except that in the former, *exp1* is only evaluated  
once. The order of evaluation of the operands is unspecified. Both oper-  
ands must either have arithmetic type or the right may have integer type  
if the left is a pointer to an object. The left operand must be a modi-  
fiable lvalue. The type of the result is the type of *exp1*. This operator  
associates right to left. *See also* assignment operator, compound.

---

Entry names are set in bold. All language elements (such as identifiers,  
operators, and preprocessor directives) and program fragments are set in a  
constant width font. Where general language syntax is shown, the parts to  
be supplied by the programmer are set in italics. In all other cases, italics are  
used for emphasis.

Many entries refer directly to other entries. For example, the above entry  
refers to “assignment,” “associativity,” “modifiable lvalue,” “operator,” and

“pointer,” among others. I deliberately chose *not* to set such references in some special font because there are so many of them; many entries would become unreadable due to the continual font changes that would be needed. The rule, then, is that every term used in an entry is either specific to C and is described in its own entry, or it is a general computing term that the reader should already know. Related entries that are *not* otherwise referenced in an entry are explicitly identified at the end of that entry, as in “*See also* assignment operator, compound.”

A number of entries, as well as names in various tables, have superscripts as in `CHAR_MAX`<sup>C89</sup>, `and`<sup>C95</sup>, and `restrict`<sup>C99</sup>. Such a superscript indicates that entry or name was introduced in the C standard or amendment produced in the corresponding year. For more information on these versions, see the entry of the same name. All entries or names nor having such a superscript are deemed to have been in general use in C implementations prior to the first C standard in 1989.

Many entries describe the library functions. Ordinarily, the entry name for a library function is an identifier, such as `isupper` or `printf`. As numerous floating-point functions come in groups of three, as in `acos`, `acosf`, and `acosl`, supporting `double`, `float`, and `long double` precisions, respectively, they are described in one entry using the following format:

---

`acos[f|l]` A function that computes the arc cosine of its argument `x` and returns a value in the range  $[0, \pi]$  radians.

```
#include <math.h>
double acos(double x);
float acosf(float x);
long double acosl(long double x);
```

If the argument is not in the range  $[-1, +1]$ , a domain error occurs.

The `float` and `long double` versions were an invention of C89, where they were optional; however, in C99, they are required.

---

Each library entry begins with a short description of that function’s purpose. This is followed by the name of the header(s) to be used to gain proper access to that function, and a prototype for that function. That is followed by the details of using that function and any related information.

C99 introduced the type qualifier keyword `restrict` and allowed it to be used in function prototypes. It was also added to numerous existing standard library functions, for example,

---

`fgets` A function ...

```
#include <stdio.h>
char *fgets(char * restrict s, int n,
            FILE * restrict stream);
```

...

---

Of course, if you are using a pre-C99 implementation, this keyword will not be supported, nor will it be present in any library function declarations.

## Operators, Punctuators, etc.

<i>Token</i>	<i>Dictionary Entry</i>
!	logical negation operator
!=	inequality operator
()	cast operator declarator, punctuation in function call operator parentheses punctuator
*	declarator, punctuation in indirection operator multiplication operator
*=	multiplication assignment operator
+	addition operator unary plus operator
++	increment operator
+=	addition assignment operator
,	comma operator comma punctuator
-	subtraction operator unary minus operator
--	decrement operator
-=	subtraction assignment operator
->	arrow operator
.	dot operator radix point
...	ellipsis punctuator
/	division operator
/* */	comment
//	comment
/=	division assignment operator
:	colon
digraph	
;	semicolon punctuator
<	less-than operator
<%	digraph
<:	digraph
<<	left-shift operator
<<=	left-shift assignment operator
<=	less-than-or-equal-to operator
=	assignment operator, simple equals punctuator
==	equality operator
>	greater-than operator
>=	greater-than-or-equal-to operator
>>	right-shift operator

## Operators, Punctuators, etc., (cont'd)

<i>Token</i>	<i>Dictionary Entry</i>
>>=	right-shift assignment operator
?:	conditional operator
?? <i>x</i>	trigraph
[ ]	declarator, punctuation in subscript operator
#	null preprocessing directive stringize operator
##	token pasting operator
%	conversion specifier remainder operator
%%	digraph
%%:	digraph
%>	digraph
%=	remainder assignment operator
&	address-of operator AND operator, bitwise
&&	AND operator, logical
&=	AND assignment operator, bitwise
\	backslash
\"	double quote escape sequence
\'	single quote escape sequence
\0	null character
\?	question mark escape sequence
\\	backslash escape sequence
\a	alert escape sequence
\b	backspace escape sequence
\ddd	octal escape sequence
\f	form-feed escape sequence
\n	new-line escape sequence
\r	carriage-return escape sequence
\t	horizontal-tab escape sequence
\U	universal character name
\u	universal character name
\v	vertical-tab escape sequence
\xhh	hexadecimal escape sequence
^	OR operator, bitwise exclusive
^=	OR assignment operator, bitwise exclusive
_	underscore
{ }	braces
	OR operator, bitwise inclusive
=	OR assignment operator, bitwise inclusive
	OR operator, logical
~	complement operator
<i>op</i> =	assignment operator, compound