## NAME

cpp − ANSI/ISO C compiler preprocessor component

## SYNOPSIS

**cpp** [ −**D***name* ] [ −**D***name*=*value* ] [ −**F***value* ] [ −**g** ] [ −**I***path* ] [ −**l** ] [ −**M** ] [ −**N** ] [ −**O** ] [ −**U***name* ] [ −**v** ] [ −**V** ] [ − **+** ] [ *infile* [ *outfile* ] ]

If the input and output filenames are omitted, they default to the standard input and output, respectively.

## DESCRIPTION

**cpp** is a preprocessor for the C and C++ programming languages conforming strictly to the 1989, 1990, 1998, and 1999 ANSI and ISO Standards for those languages, and is a component of the **lcc**(1) compiler system.

**cpp** filters an input stream, obeying preprocessing directives, and copying nondirective text to an output stream, after expanding any macros that are recognized in that text. Text that matches the comment syntax of the programming language is discarded. Nondirective text which does not require processing is simply copied verbatim. Handling of binary data is undefined, and unpredictable.

Because the C preprocessor is not normally invoked directly by humans, it is seldom found in the default **PATH**. To see where **cpp** is installed, run a C or C++ compiler with an option to display the commands that it invokes. This option is often **−v** (as with **lcc**(1)), or **−#**.

## OPTIONS

**cpp** recognizes the following command-line options:

**−D***name* Define *name* to have the integer value 1.

**−D***name*=*value* Define *name* to have the string value *value*. If the value is a character string, then quotation marks, suitably protected against shell interpretation, must be explicitly provided: compare **−D***version=23* with **−D***date='"29 March 2001"'*. **−F***value* This option is recognized, for historical reasons, but is ignored.

**−g** This option is recognized, for historical reasons, but is ignored.

**−I***path* Add *path* to the end of the list of directories to be searched for included files. **cpp** has *no* predefined default directories, so compilers that invoke **cpp** are expected to pass it their own list of include directories, prefixed by any user-defined directories. Directories are always searched in the order given by **−I** options.

**−l** This option is recognized, for historical reasons, but is ignored.

**−M** Suppress normal preprocessor output, and produce *Makefile* object file dependencies instead.

If **−M** is given at least once, generate a list of dependencies on quoted header files (**#include** *"myfile.h"*), and write them on the standard output unit.

If **−M** is given more than once, the output also includes dependencies on angle-bracketed header files (**#include** <*sysfile.h*>).

**−N** Do not search *any* of the directories currently in the search list for include files. Only those directories specified by subsequent explicit **−I** options will be searched, in the order given.

**−O** This option is recognized, for historical reasons, but is ignored.

**−U***name* Undefine any prior definition of *name*. It is not an error for the name to already be undefined, and no diagnostic is issued in either case.

Certain special names are required by the language Standards to be immune to user redefinition, once defined, although **cpp** itself may alter their values: **__DATE__**, **__FILE__**, **__LINE__**, **__STDC__**, **__TIME__**, and **defined**.

**−v** Display the **cpp** version number and revision date on the standard error unit, and continue processing.

**−V** Write token parsing progress reports on the standard error unit.

**−+** Recognize C++ (and 1999 Standard C) comments, from // up to, but not including, the next end-of-line. The current version of **cpp** used by **lcc**(1) always has this flag set, because some pre-1999 Standard C compilers supported C++-style comments as an extension.

## USAGE

The precise details of the preprocessing actions are complex, and documented only briefly here. Consult one of the language Standards, or a good textbook about C or C++, for more information.

The recognized directives are: **#define**, **#elif**, **#else**, **#endif**, **#error**, **#eval**, **#if**, **#ifdef**, **#ifndef**, **#include**, **#line**, **#pragma**, and **#undef**. Such directives must occur at the start of a line, although whitespace may optionally precede and/or follow the initial sharp (**#**).

All other input that resembles a directive, such as **#ident**, **#import**, **#include_next** <*filename.h*>, or **#module**, is ignored, with a warning, and copied to the output stream.

The syntax of *constant-expression* used in the descriptions below matches that of Standard C and C++, excluding assignment operators, and has the same precedence rules.

Arithmetic is carried out as if all values are of type **long int** or **unsigned long int**, but the sign of character constants is implementation-defined.

Constructs like the C/C++ **sizeof** operator, typecasts, and **enum**s, floating-point arithmetic, and character-string matching, are *not* available in preprocessor expressions.

**defined** name

**defined**(name) This directive is used in constant expressions in conditional statements: it evaluates to one if *name* is defined, and zero if not.

**#define** *name* Define *name* to expand to 1.

**#define** *name value* Define *name* to expand to *value*.

**#define** *name(arg1,arg2,...,argn) value*

The parenthesized list may contain zero or more named arguments, using the normal C/C++ identifier syntax (initial letter or underscore, followed by any number of letters, digits, and underscores). An occurrence of *name(rep1,rep2,...,repn)* in the input stream is replaced by the expansion of *value*, where each occurrence of *arg1*, *arg2*, ... *argn* is replaced by the corresponding text *rep1*, *rep1*, ... *repn*, without further substitution. After that expansion is complete, it is rescanned for further macro substitutions.

In order for the expansion to take place, the input stream must contain *name*, optionally followed by whitespace, and then the argument list. In particular, if the input stream contains *(name)(v1,v2,...,vn)*, then macro expansion does *not* occur. This can be used to protect against such expansion: *foo(a,b,c)* might be expanded, but *(foo)(a,b,c)* is never expanded (although the arguments will be). Both forms are valid, and equivalent, C and C++ code.

**#elif** *constant-expression*

If the *constant-expression* is nonzero, and no prior expression in preceding **#if** *constant-expression* or **#elif** *constant-expression* parts of this conditional statement has evaluated to nonzero, expand the following lines down to, but excluding, the next matching **#elif** *constant-expression*, **#else**, or **#endif**, and then skip to the statement following the matching **#endif**.

Nested conditionals are permitted, provided each matching **#if**/**#endif** pair is contained within the same source file.

**#else** If no *constant-expression* in preceding **#if** *constant-expression* or **#elif** *constant-expression* parts of this conditional statement has evaluated to nonzero, expand the

following lines down to, but excluding, the next matching **#endif**, and then skip to the statement following that **#endif**.

**#endif** Terminate the most-recently opened conditional **#if** *constant-expression***.**

**#error** *message text* Issue an error message to the standard error unit with the specified text, after normal preprocessing expansion of the text. This does *not* affect the final status return code from **cpp**.

**#eval** *constant-expression***.**
This directive evaluates the expression, like **#if** or **#elif**, and then discards the result.

**#eval** is an extension to the C and C++ Standards, and it is unclear why it even exists, since, in the absence of assignment operators, the result of the evaluation cannot be detected, or have any side effects!

**#if** *constant-expression*
If the *constant-expression* is nonzero, expand the following lines down to, but excluding, the next matching **#elif** *constant-expression***, #else**, or **#endif**, and then skip to the statement following the matching **#endif**.

Nested conditionals are permitted, provided each matching **#if**/**#endif** pair is contained within the same source file.

**#ifdef** *name* Act like **#if**, with the expression evaluating to nonzero if *name* is defined, and zero if it is not.

This can be coded equivalently as **#if defined***(name)*.

**#ifndef** *name* Act like **#if**, with the expression evaluating to nonzero if *name* is not defined, and zero if it is.

This can be coded equivalently as **#if !defined***(name)*.

**#include** *"filename"* Temporarily divert input processing to the named file, which must be found either in the directory of the file containing that directive, or in directories named by **−I***path* options that define the include directory search path.

While it is conventional to name such header files with a *.h* extension, it is not necessary to do so. Indeed, Standard C++ has many header files that do not end with *.h*.

Include files may be nested to a maximum depth of 10.

**#include** <*filename*>
Temporarily divert input processing to the named file, which must be found in directories named by **−I***path* options that define the include directory search path.

While it is conventional to name such header files with a *.h* extension, it is not necessary to do so. All of those defined in Standard C have such extensions.

Include files may be nested to a maximum depth of 10.

**#** *integer-constant*
**#line** *integer-constant*
**#** *integer-constant "filename"*
**#line** *integer-constant "filename"*
Generate line number information for subsequent programs. The constant is interpreted as the line number of the next line, and *"filename"* as the file from which it came. It *"filename"* is omitted, the current filename is unchanged.

Input **#** and **#line** directives are copied verbatim to the output stream. New ones are generated at suitable points in the input stream, such as at the start of each file, and the resumption of input after completion of **#include** file processing.

**#pragma** *text* This directive is copied verbatim from the input stream to the output stream, after normal preprocessing expansion of the text.

The effect of **#pragma** is implementation defined. It is thus system-dependent, and rarely useful for end users, because the same pragma might be interpreted in widely different ways by different compilers, or accepted by one, and diagnosed as erroneous by others.

**rcc**(1) recognizes just one form of this directive, **#pragma ref** *identifier***,** but currently, no use is made of it by the **lcc**(1) compiler system.

**#undef** *name* Undefine any prior definition of *name*. It is not an error for the name to already be undefined, and no diagnostic is issued in either case.

## ENVIRONMENT VARIABLES

**cpp** does not use any environment variables.

## FILES

**cpp** opens only those files explicitly given on its command line.

## SEE ALSO

**CC**(1), **c89**(1), **cc**(1), **cpp**(1), **g++**(1), **gcc**(1), **lcc**(1), **pgCC**(1), **pgcc**(1), **rcc**(1).

C. W. Fraser and D. R. Hanson, *A Retargetable C Compiler: Design and Implementation*, Addison-Wesley, 1995. ISBN 0-8053-1670-1.

The World-Wide Web page at `http://www.cs.princeton.edu/software/lcc/`.

S. P. Harbison and G. L. Steele, Jr., *C: A Reference Manual*, 4th ed., Prentice-Hall, 1995.

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed., Prentice-Hall, 1988.

American National Standards Inst., *American National Standard for Information Systems—Programming Language—C*, ANSI X3.159-1989, New York, 1990.

International Organization for Standardization, *ISO/IEC 9899:1990: Programming languages — C*, Geneva, Switzerland, 1990.

## BUGS

Mail bug reports along with the shortest program that exposes them, and the details reported by **cpp**'s **−v** option, to `lcc-bugs@princeton.edu`. The World-Wide Web page at URL `http://www.cs.princeton.edu/software/lcc/` includes detailed instructions for reporting bugs.