NAME

cpp – the C language preprocessor

SYNOPSIS

/usr/ccs/lib/cpp [**–BCHMpPRT**] [**–undef**] [**–D***name*] [**–D***name***=***def*] [**–I***directory*]
[**–U***name*] [**–Y***directory*] [*input-file* [*output-file*]]

AVAILABILITY

SUNWsprot

DESCRIPTION

cpp is the C language preprocessor. It is invoked as the first pass of any C compilation started with the **cc**(1B) command; however, **cpp** can also be used as a first-pass preprocessor for other Sun compilers.

Although **cpp** can be used as a macro processor, this is not normally recommended, as its output is geared toward that which would be acceptable as input to a compiler's second pass. Thus, the preferred way to invoke **cpp** is through the **cc**(1B) command, or some other compilation command. For general-purpose macro-processing, see **m4**(1), and the chapter on **m4** in *Programming Utilities Guide*.

cpp optionally accepts two filenames as arguments. *input-file* and *output-file* are, respectively, the input and output files for the preprocessor. They default to the standard input and the standard output.

OPTIONS

The following options are supported:

–B Support the C++ comment indicator '**//**'. With this indicator everything on the line after the **//** is treated as a comment.

–C Pass all comments (except those that appear on **cpp** directive lines) through the preprocessor. By default, **cpp** strips out C-style comments.

–H Print the pathnames of included files, one per line on the standard error.

–M Generate a list of makefile dependencies and write them to the standard output. This list indicates that the object file which would be generated from the input file depends on the input file as well as the include files referenced.

–p Use only the first eight characters to distinguish preprocessor symbols, and issue a warning if extra tokens appear at the end of a line containing a directive.

–P Preprocess the input without producing the line control information used by the next pass of the C compiler.

–R Allow recursive macros.

–T Use only the first eight characters for distinguishing different preprocessor names. This option is included for backward compatibility with systems which always use only the first eight characters.

–undef Remove initial definitions for all predefined symbols.

–D*name* Define *name* as 1 (one). This is the same as if a **–D***name***=1** option appeared on the **cpp** command line, or as if a

#define *name* **1**

line appeared in the source file that **cpp** is processing.

–**D***name***=***def* Define *name* as if by a **#define** directive. This is the same as if a

#define *name* **def**

line appeared in the source file that **cpp** is processing. The –**D** option has lower precedence than the –**U** option. That is, if the same name is used in both a –**U** option and a –**D** option, the name will be undefined regardless of the order of the options.

–**I***directory* Insert *directory* into the search path for **#include** files with names not beginning with '**/**'. *directory* is inserted ahead of the standard list of ''include'' directories. Thus, **#include** files with names enclosed in double-quotes (**"**) are searched for first in the directory of the file with the **#include** line, then in directories named with –**I** options, and lastly, in directories from the standard list. For **#include** files with names enclosed in angle-brackets (**< >**), the directory of the file with the **#include** line is not searched. See **Details** below for exact details of this search order.

–**U***name* Remove any initial definition of *name*, where *name* is a symbol that is predefined by a particular preprocessor. Here is a partial list of symbols that may be predefined, depending upon the architecture of the system:

Operating System:	**ibm**, **gcos**, **os**, **tss** and **unix**
Hardware:	**interdata**, **pdp11**, **u370**, **u3b**, **u3b2**, **u3b5**, **u3b15**, **u3b20d**, **vax**, **ns32000**, **iAPX286**, **i386**, **sparc**, and **sun**
UNIX system variant:	**RES**, and **RT**
The **lint** command:	**lint**

The symbols **sun**, **sparc** and **unix** are defined for all Sun systems.

–**Y***directory* Use directory *directory* in place of the standard list of directories when searching for **#include** files.

USAGE

Directives

All **cpp** directives start with a hash symbol (**#**) as the first character on a line. White space (SPACE or TAB characters) can appear after the initial **#** for proper indentation.

#define *name token-string*
Replace subsequent instances of *name* with *token-string*.

#define *name***(***argument* [**,** *argument*] . . . **)** *token-string*
There can be no space between *name* and the '**(**'. Replace subsequent instances of *name*, followed by a parenthesized list of arguments, with *token-string*, where each occurrence of an *argument* in the *token-string* is replaced by the corresponding token in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, **cpp** re-starts its scan for names to

expand at the beginning of the newly created *token-string*.

#undef *name*

Remove any definition for the symbol *name*. No additional tokens are permitted on the directive line after *name*.

#include "*filename***"**
#include <*filename***>**

Read in the contents of *filename* at this location. This data is processed by **cpp** as if it were part of the current file. When the **<***filename***>** notation is used, *filename* is only searched for in the standard ''include'' directories. See the –**I** and –**Y** options above for more detail. No additional tokens are permitted on the directive line after the final '**"**' or '**>**'.

#line *integer-constant* **"***filename***"**

Generate line control information for the next pass of the C compiler. *integer-constant* is interpreted as the line number of the next line and *filename* is interpreted as the file from where it comes. If **"***filename***"** is not given, the current filename is unchanged. No additional tokens are permitted on the directive line after the optional *filename*.

#if *constant-expression*

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** directive, appear in the output only if *constant-expression* yields a nonzero value. All binary non-assignment C operators, including '**&&**', '**| |**', and '**,**', are legal in *constant-expression*. The '**?:**' operator, and the unary '–', '**!**', and '**~**' operators, are also legal in *constant-expression*.

The precedence of these operators is the same as that for C. In addition, the unary operator **defined**, can be used in *constant-expression* in these two forms: '**defined (** *name* **)**' or '**defined** *name*'. This allows the effect of **#ifdef** and **#ifndef** directives (described below) in the **#if** directive. Only these operators, integer constants, and names that are known by **cpp** should be used within *constant-expression*. In particular, the **size of** operator is not available.

#ifdef *name*

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has been defined, either with a **#define** directive or a –**D** option, and in the absence of an intervening **#undef** directive. Additional tokens after *name* on the directive line will be silently ignored.

#ifndef *name*

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has *not* been defined, or if its definition has been removed with an **#undef** directive. No additional tokens are permitted on the directive line after *name*.

#elif *constant-expression*

Any number of **#elif** directives may appear between an **#if**, **#ifdef**, or **#ifndef** directive and a matching **#else** or **#endif** directive. The lines following the **#elif** directive appear in the output only if all of the following conditions hold:

- The *constant-expression* in the preceding **#if** directive evaluated to zero, the *name* in the preceding **#ifdef** is not defined, or the *name* in the preceding **#ifndef** directive *was* defined.
- The *constant-expression* in all intervening **#elif** directives evaluated to zero.
- The current *constant-expression* evaluates to non-zero.

If the *constant-expression* evaluates to non-zero, subsequent **#elif** and **#else** directives are ignored up to the matching **#endif**. Any *constant-expression* allowed in an **#if** directive is allowed in an **#elif** directive.

#else This inverts the sense of the conditional directive otherwise in effect. If the preceding conditional would indicate that lines are to be included, then lines between the **#else** and the matching **#endif** are ignored. If the preceding conditional indicates that lines would be ignored, subsequent lines are included in the output. Conditional directives and corresponding **#else** directives can be nested.

#endif End a section of lines begun by one of the conditional directives **#if**, **#ifdef**, or **#ifndef**. Each such directive must have a matching **#endif**.

Macros

Formal parameters for macros are recognized in **#define** directive bodies, even when they occur inside character constants and quoted strings. For instance, the output from:

```
#define abc(a)  |`|a|
abc(xyz)
```

is:

```
# 1 ""

|`|xyz |
```

The second line is a NEWLINE. The last seven characters are ‘‘**|`|xyz |**’’ (vertical-bar, backquote, vertical-bar, x, y, z, vertical-bar). Macro names are not recognized within character constants or quoted strings during the regular scan. Thus:

```
#define abc  xyz
printf("abc");
```

does not expand **abc** in the second line, since it is inside a quoted string that is not part of a **#define** macro definition.

Macros are not expanded while processing a **#define** or **#undef**. Thus:

```
#define abc zingo
#define xyz abc
#undef abc
xyz
```

produces **abc**. The token appearing immediately after an **#ifdef** or **#ifndef** is not expanded.

Macros are not expanded during the scan which determines the actual parameters to another macro call. Thus:

```
#define reverse(first,second)second first
#define greeting hello
reverse(greeting,
#define greeting goodbye
)
```

produces '' **#define hello goodbye hello**''.

Output

Output consists of a copy of the input file, with modifications, plus lines of the form:

> #*lineno* " *filename* " "*level* "

indicating the original source line number and filename of the following output line and whether this is the first such line after an include file has been entered (*level*=**1**), the first such line after an include file has been exited (*level*=**2**), or any other such line (*level* is empty).

Details

Directory Search Order

#include files are searched for in the following order:

1. The directory of the file that contains the **#include** request (that is, **#include** is relative to the file being scanned when the request is made).
2. The directories specified by –**I** options, in left-to-right order.
3. The standard directory(s) (**/usr/include** on UNIX systems).

Special Names

Two special names are understood by **cpp**. The name **_ _LINE_ _** is defined as the current line number (a decimal integer) as known by **cpp**, and **_ _FILE_ _** is defined as the current filename (a C string) as known by **cpp**. They can be used anywhere (including in macros) just as any other defined name.

Newline Characters

A NEWLINE character terminates a character constant or quoted string. An escaped NEWLINE (that is, a backslash immediately followed by a NEWLINE) may be used in the body of a **#define** statement to continue the definition onto the next line. The escaped NEWLINE is not included in the macro value.

Comments

Comments are removed (unless the –**C** option is used on the command line). Comments are also ignored, except that a comment terminates a token.

EXIT STATUS

The following exit values are returned:

0	Successful completion.
non-zero	An error occurred.

SEE ALSO

cc(1B), **m4**(1)

Programming Utilities Guide

DIAGNOSTICS

The error messages produced by **cpp** are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NOTES

When NEWLINE characters were found in argument lists for macros to be expanded, some previous versions of **cpp** put out the NEWLINE characters as they were found and expanded. The current version of **cpp** replaces them with SPACE characters.

Because the standard directory for included files may be different in different environments, this form of **#include** directive:

#include <file.h>

should be used, rather than one with an absolute path, like:

#include "/usr/include/file.h"

cpp warns about the use of the absolute pathname.

While the compiler allows 8-bit strings and comments, 8-bits are not allowed anywhere else.