

The C Preprocessor

For GCC version 16.0.0 (pre-release)

(GCC)

Richard M. Stallman, Zachary Weinberg

Copyright © 1987-2025 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation. A copy of the license is included in the section entitled “GNU Free Documentation License”.

This manual contains no Invariant Sections. The Front-Cover Texts are (a) (see below), and the Back-Cover Texts are (b) (see below).

(a) The FSF’s Front-Cover Text is:

A GNU Manual

(b) The FSF’s Back-Cover Text is:

You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development.

GNU Free Documentation License	72
ADDENDUM: How to use this License for your documents	79
Index of Directives	80
Option Index	80
Concept Index	82

preprocessing number. When pasted, they make a longer identifier. This isn't the only valid case. It is also possible to concatenate two numbers (or a number and a name, such as 1.5 and e3) into a number. Also, multi-character operators such as += can be formed by token pasting.

However, two tokens that don't together form a valid token cannot be pasted together. For example, you cannot concatenate x with + in either order. If you try, the preprocessor issues a warning and emits the two tokens. Whether it puts white space between the tokens is undefined. It is common to find unnecessary uses of '##' in complex macros. If you get this warning, it is likely that you can simply remove the '##'.

Both the tokens combined by '##' could come from the macro body, but you could just as well write them as one token in the first place. Token pasting is most useful when one or both of the tokens comes from a macro argument. If either of the tokens next to an '##' is a parameter name, it is replaced by its actual argument before '##' executes. As with stringizing, the actual argument is not macro-expanded first. If the argument is empty, that '##' has no effect.

Keep in mind that the C preprocessor converts comments to whitespace before macros are even considered. Therefore, you cannot create a comment by concatenating '/' and '*'. You can put as much whitespace between '##' and its operands as you like, including comments, and you can put comments in arguments that will be concatenated. However, it is an error if '##' appears at either end of a macro body.

Consider a C program that interprets named commands. There probably needs to be a table of commands, perhaps an array of structures declared as follows:

```
struct command
{
    char *name;
    void (*function) (void);
};

struct command commands[] =
{
    { "quit", quit_command },
    { "help", help_command },
    ...
};
```

It would be cleaner not to have to give each command name twice, once in the string constant and once in the function name. A macro which takes the name of a command as an argument can make this unnecessary. The string constant can be created with stringizing, and the function name by concatenating the argument with '_command'. Here is how it is done:

```
#define COMMAND(NAME) { #NAME, NAME ## _command }

struct command commands[] =
{
    COMMAND (quit),
    COMMAND (help),
    ...
};
```


