C/C++ macro string concatenation

Asked 12 years, 10 months ago Modified 2 years, 10 months ago Viewed 296k times

173

```
#define STR1      "s"
#define STR2      "1"
#define STR3      STR1 ## STR2
```

Is it possible to concatenate `STR1` and `STR2`, to `"s1"`? You can do this by passing args to another Macro function. But is there a direct way?

c++ c c-preprocessor

Share Follow

edited Feb 22, 2021 at 15:08
Neuron
5,325 5 40 61

asked Mar 10, 2011 at 6:36
tvr
4,475 9 24 29

Shouldn't it be #define STR3 STR1 ## STR2 – Shrinidhi Mar 10, 2011 at 6:39

1 It shouldn't be either because that defines STR3 to be the preprocessing token STR1STR2. And passing args to another macro function doesn't help, because string literals can't be pasted together -- "s""1" is not a valid token. – Jim Balter Mar 10, 2011 at 7:09

3 Answers

Sorted by: Highest score (default)

225

If they're both strings you can just do:

```
#define STR3 STR1 STR2
```

This then expands to:

```
#define STR3 "s" "1"
```

and in the C language, separating two strings with space as in `"s" "1"` is exactly equivalent to having a single string `"s1"`.

Share Follow

edited Nov 3, 2020 at 8:12

answered Mar 10, 2011 at 6:53

Ciro Santilli OurBigBook.com **357k** 110 1225 1006

Sean **5,340** 2 24 21

25 Technically string concatenation is done at the language level. – Martin York Mar 10, 2011 at 6:59

58 The preprocessor does no such thing. It's the C language proper that treats adjacent string literals as if they were a single string literal. – Jim Balter Mar 10, 2011 at 7:00

11 It's more than a technicality - you can't concatenate `L"a"` and `"b"` to get `L"ab"`, but you *can* concatenate `L"a"` and `L"b"` to get `L"ab"`. – MSalters Mar 10, 2011 at 8:59

1 This does not work if you try `#include STR3` with `STR3` being a valid header file. Does anyone know how to? – Zythos Nov 11, 2020 at 16:37

2 @Zythos - you might want to post a separate question with some details about what you are trying to do, what you expect to happen, and what actually happens. – Sean Nov 12, 2020 at 17:15

147

You don't need that sort of solution for string literals, since they are concatenated at the language level, and it wouldn't work anyway because "s""1" isn't a valid preprocessor token.

[Edit: In response to the incorrect "Just for the record" comment below that unfortunately received several upvotes, I will reiterate the statement above and observe that the program fragment

```
#define PPCAT_NX(A, B) A ## B
PPCAT_NX("s", "1")
```

produces this error message from the preprocessing phase of gcc: *error: pasting ""s"" and ""1"" does not give a valid preprocessing token*

]

However, for general token pasting, try this:

```
/*
 * Concatenate preprocessor tokens A and B without expanding macro definitions
 * (however, if invoked from a macro, macro arguments are expanded).
 */
#define PPCAT_NX(A, B) A ## B

/*
 * Concatenate preprocessor tokens A and B after macro-expanding them.
 */
#define PPCAT(A, B) PPCAT_NX(A, B)
```

Then, e.g., both `PPCAT_NX(s, 1)` and `PPCAT(s, 1)` produce the identifier `s1`, unless `s` is defined as a macro, in which case `PPCAT(s, 1)` produces `<macro value of s>1`.

Continuing on the theme are these macros:

```
/*
 * Turn A into a string literal without expanding macro definitions
 * (however, if invoked from a macro, macro arguments are expanded).
 */
#define STRINGIZE_NX(A) #A

/*
 * Turn A into a string literal after macro-expanding it.
 */
#define STRINGIZE(A) STRINGIZE_NX(A)
```

Then,

```
#define T1 s
#define T2 1
STRINGIZE(PPCAT(T1, T2)) // produces "s1"
```

By contrast,

```
STRINGIZE(PPCAT_NX(T1, T2)) // produces "T1T2"
STRINGIZE_NX(PPCAT_NX(T1, T2)) // produces "PPCAT_NX(T1, T2)"

#define T1T2 visit the zoo
STRINGIZE(PPCAT_NX(T1, T2)) // produces "visit the zoo"
STRINGIZE_NX(PPCAT(T1, T2)) // produces "PPCAT(T1, T2)"
```

Share Follow

edited Aug 8, 2019 at 21:56

answered Mar 10, 2011 at 7:03

Jim Balter
16.2k 3 44 67

8 Just for the record, `"s""1"` is valid in C (and C++). They are two tokens (string literals) that the compiler would concat itself and threat as one token. – Shahbaz Jul 31, 2012 at 9:24

6 You misunderstand both my comment and the C language. I said `"s""1" isn't a valid token` -- that is correct; it is, as you say, *two* tokens. But tacking them together with ## would make them a *single* preprocessing token, not two tokens, and so the compiler would not do a concatenation, rather the lexer would reject them (the language requires a diagnostic). – Jim Balter Jul 31, 2012 at 9:30

8 @mr5 Read the comments, carefully. Macro names passed as macro arguments are not expanded before being passed. They are, however, expanded in the body of the macro. So if A is defined as FRED, STRINGIZE_NX(A) expands to "A" but STRINGIZE(A) expands to STRINGIZE_NX(FRED) which expands to "FRED". – Jim Balter Jul 21, 2014 at 19:25

1 @bharath *the resulting string is "PPCAT(T1,T2)"* -- as expected and desired. *and not the expected "s1"* -- not expected at all. *Why do we need an extra indirection/nesting?* -- Read the code comments, and my comment above with the 6 upvotes. Only the bodies of macros are expanded; outside of macro bodies, macro arguments between parentheses are *not* expanded before being passed to macros. So `STRINGIZE_NX(whatever occurs here)` expands to "whatever occurs here", regardless of any macro definitions for whatever, occurs, or here. – Jim Balter Jan 10, 2018 at 23:42

1 @bharath Of course it doesn't print "Name A" -- A is the parameter name, not the argument to the macro, which is ALEX. You claimed `if A is defined as FRED then STRINGIZE_NX(A) still expands to "FRED"` -- that is false, and is nothing like your test. You're trying hard not to understand or get this right, and I'm not going to respond to you further. – Jim Balter Jan 11, 2018 at 19:23

28

Hint: The `STRINGIZE` macro above is cool, but if you make a mistake and its argument isn't a macro - you had a typo in the name, or forgot to `#include` the header file - then the compiler will happily put the purported macro name into the string with no error.

If you intend that the argument to `STRINGIZE` is always a macro with a normal C value, then

```
#define STRINGIZE(A) ((A),STRINGIZE_NX(A))
```

will expand it once and check it for validity, discard that, and then expand it again into a string.

It took me a while to figure out why `STRINGIZE(ENOENT)` was ending up as `"ENOENT"` instead of `"2"` ... I hadn't included `errno.h`.

Share Follow

answered Feb 8, 2012 at 0:34

Jordan Brown
374 3 4

3 Important observation, and +1 for proper use of the `,` operator. :) – Jesse Chisholm Sep 21, 2015 at 21:20

2 There's no particular reason why the content of the string should be a valid C expression. If you want to do this, I advise giving it a different name, like STRINGIZE_EXPR. – Jim Balter Oct 26, 2018 at 14:01

That trick may have worked in isolation. But it prevents the compiler from seeing a sequence of strings which it will concatenate. (resulting in sequences like `((1),"1") "." ((2),"2")` instead of just "1" "." "2") – automorphic Mar 6, 2020 at 7:17

2 Just to clarify what automorphic is saying: with the original `STRINGIZE` definition, `"The value of ENOENT is " STRINGIZE(ENOENT)` works, whereas `"The value of ENOENT is" STRINGIZE_EXPR(X)` produces an error. – Jim Balter Jun 25, 2020 at 21:53

In the above comment I meant `STRINGIZE_EXPR(ENOENT)` rather than `STRINGIZE_EXPR(X)`. – Jim Balter Jul 13, 2023 at 6:27