Passing variable name as string to function with default parameters

Asked 9 years, 6 months ago Active 9 years, 6 months ago Viewed 5k times

4

1

Let's say there is a debugging function, simplified here as:

```
void DumpString(char* var, char* varname) {
    printf("%s : '%s'\n", varname, var);
}

char str[10]="foobar";
DumpString(str, "str");

> str : foobar
```

Let's make it easier by removing the unnecessarily extraneous requirement of passing the variable twice, once in quotes:

```
#define VARASSTR(v) #v

void DumpString(char* var) {
    printf("%s : '%s'\n", VARASSTR(var), var);
}

char str[10]="foobar";
DumpString(str);

> var : foobar
```

Oops! It uses the local variable name instead of the one passed in. Let's try a different (and less ideal) tack:

```
#define DumpStr(v) DumpString(v, #v)

void DumpString(char* var, char* varname) {
    printf("%s : '%s'\n", varname, var);
}

char str[10]="foobar";
DumpStr(str);

> str : foobar
```

Great it works. But what if the function was a little more complicated:

```
void DumpString(char* var, char* varname, int optionalvar=0) {
    printf("%s : '%s'\n", varname, var);
    printf("blah: %d", optionalvar);
}
```

It is not possible to overload a macro, so `DumpStr` won't work, and we have already ruled out the version with `VARASSTR`.

How can this be handled (without resorting to multiple similarly, but differently-named functions/macros)?

c++ stringify variable-names default-parameters

Share Follow

edited Apr 2 '12 at 1:25

asked Apr 2 '12 at 0:08

Synetech

9,057 ● 8 ● 60 ● 91

"How can this be handled (without resorting to multiple similarly, but differently-named functions/macros)?" It can't. You can't overload macros. – meagar ♦ Apr 2 '12 at 0:11

You should try some language with reflection capabilities. No, really. – iehrlich Apr 2 '12 at 0:13

1 You could use Variadic macro trick to "simulate" overloading with macros. – Jesse Good Apr 2 '12 at 0:27

Thanks for linking my question. :-) – R.. GitHub STOP HELPING ICE Apr 2 '12 at 1:15

You cannot overload functions too (or why did you tag this question `c`), so what is the problem? – asaelr Apr 2 '12 at 1:18

1 Answer

Active Oldest Votes

1

This is non-standard, but works as an extension in GNU C:

```
#define DumpStr(v, ...) DumpString(v, #v, ##__VA_ARGS__)
```

In GNU C, you can pass no arguments to a variadic macro, and the "token pasting operator" `##` when applied between a comma and an empty variadic argument list produces nothing (so the trailing comma is suppressed).

In Visual C++, I believe the token pasting operator `##` is unnecessary (and will probably break the macro), as Visual C++ automatically suppresses a trailing comma if it appears before an empty variadic argument list.

Note that the only thing that makes this nonstandard is the desire to sometimes pass an empty argument list. Variadic macros are standardized in both C99 and C++11.

Edit: And here's an example that doesn't use non-standard features. You can see why some people really, really wish this sort of thing was addressed in the standard:

```
#define DUMPSTR_1(v) DumpString(v, #v)
#define DUMPSTR_2(v, opt) DumpString(v, #v, opt)
#define DUMPSTR_NARG(...) DUMPSTR_ARG_N(__VA_ARGS__, 4, 3, 2, 1, 0)
#define DUMPSTR_ARG_N(_1, _2, _3, _4, n, ...) n
#define DUMPSTR_NC(f, ...) f(__VA_ARGS__)
#define DUMPSTR_NB(nargs, ...) DUMPSTR_NC(DUMPSTR_ ## nargs, __VA_ARGS__)
```

```
#define DUMPSTR_NA(nargs, ...) DUMPSTR_NB(nargs, __VA_ARGS__)
#define DumpStr(...) DUMPSTR_NA(DUMPSTR_NARG(__VA_ARGS__), __VA_ARGS__)
```

There's probably a few cleaner ways to do this. But not that many.

Edit 2: And here's yet another example that doesn't use non-standard features, courtesy of R..

```
#define STRINGIFY_IMPL(s) #s
#define STRINGIFY(s) STRINGIFY_IMPL(s)
#define ARG1_IMPL(a, ...) a
#define ARG1(...) ARG1_IMPL(__VA_ARGS__, 0)
#define DumpStr(...) DumpString(STRINGIFY(ARG1(__VA_ARGS__)), __VA_ARGS__)
```

Note that this requires the argument order of the `DumpString` to be changed so that the stringified function name is the first argument.

Share Follow

edited May 23 '17 at 12:24
Community Bot
1 • 1

answered Apr 2 '12 at 0:18
John Calsbeek
34.6k • 7 • 90 • 100

There are solutions to this problem without nonstandard hacks like this. – R.. GitHub STOP HELPING ICE Apr 2 '12 at 0:30

1 In particular, you subsume the whole argument list into the `...` and then use macros that extract the first (or whatever index you want) argument from `__VA_ARGS__`. – R.. GitHub STOP HELPING ICE Apr 2 '12 at 0:32

@R.. Yes, there is. And *wow* is it terrifying. I added a self-contained way that I drummed up. Can you suggest a better one? – John Calsbeek Apr 2 '12 at 0:54

I haven't used it, but I believe boost preprocessor will hide some of that ugliness for you if you would like. – Jesse Good Apr 2 '12 at 1:01

@Jesse Yes, it will, at the cost of a large unwieldy dependency. Between that and not being standards-compliant, I'll take the one-line solution. (Assuming it works on your compiler.) – John Calsbeek Apr 2 '12 at 1:03

@R.. I can't think of any better ways to do it without running into warnings about too many or too few parameters...? – John Calsbeek Apr 2 '12 at 1:09

I was thinking something like `#define DumpStr(...) DumpString(STRINGIFY(ARG1(__VA_ARGS__)), __VA_ARGS__)` (note that the order of arguments to `DumpString` has to be changed then, but this function should presumably not be used directly anyway). – R.. GitHub STOP HELPING ICE Apr 2 '12 at 1:15

@R.. But doesn't the definition of `ARG1` run into the same problems? It might have to take an empty variadic argument list, which is non-standard, or else use a messy implementation like mine.
– John Calsbeek Apr 2 '12 at 1:16

`#define ARG1_X(a, ...) a` and `#define ARG1(...) ARG1_X(__VA_ARGS__, 0)` should work.
– R.. GitHub STOP HELPING ICE Apr 2 '12 at 1:18

@R.. Ah, yes, that's not bad. That only requires *four* supporting macros, not seven. (Somehow, in C-land, we consider that elegant.) – John Calsbeek Apr 2 '12 at 1:25

Well most of them are reusable and tend to already be handy if you're using the preprocessor a lot...
– R.. GitHub STOP HELPING ICE Apr 2 '12 at 3:26