

Re: Macro substitution conundrum

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2007-12/msg03378.html

- *From:* David Thompson <dave.thompson2@xxxxxxxxxxxx>
 - *Date:* Mon, 24 Dec 2007 00:57:14 GMT
-

On Thu, 13 Dec 2007 18:55:13 GMT, Harold Weissman
<HaroldW22@xxxxxxxxxxxx> wrote:

We also have, in some include file used by the C file in question,

```
#ifdef COMPILE_TIME_FLAG
#define MY_SYMBOL 1
#else
#define MY_SYMBOL 0
#endif
```

This #define's MY_SYMBOL to either 1 or 0 depending on the value of COMPILE_TIME_FLAG *at the point of the #include*.

What I would like is for [one use] to be assigned 1 or 0, depending on whether or not COMPILE_TIME_FLAG is defined, while y is to be assigned 0 no matter whether or not COMPILE_TIME_FLAG is defined. As you can see, this is a toy example that is trivially solved by other means; I am just illustrating a general question.

Something I tried was

```
#define NO_REPLACE

int g(void)
{
int y ;

#ifdef COMPILE_TIME_FLAG
#ifdef NO_REPLACE
#undef COMPILE_TIME_FLAG
#define SET_COMPILE_TIME_FLAG
#endif
y = MY_SYMBOL ;
#ifdef SET_COMPILE_TIME_FLAG
#define COMPILE_TIME_FLAG
```

Re: Macro substitution conundrum

```
#endif  
#endif
```

If `COMPILE_TIME_FLAG` is unset this leaves `y` uninitialized garbage; I hope you meant to put or leave something in the alternative. But as you say it doesn't accomplish what you wanted because the expansion of `MY_SYMBOL` doesn't depend on the value of `COMPILE_TIME_FLAG` **here**.

```
return y + 4 ;  
}
```

but it does not work (and it is ugly to boot.)

Any ideas as to how to do this? Actually, is it doable?

Yes, but not if you want to work for my company or sell to me.

If you want an expression-like macro `X` whose evaluated value depends on the value of another macro `Y` where `X` is used, `X` must be an expression which uses (invokes) `Y` and depends on the result. Here:

```
#ifdef COMPILE_TIME_FLAG  
#define MY_SYMBOL (NO_REPLACE? 0: 1)  
#else  
#define MY_SYMBOL (NO_REPLACE? 0: 0) /* or just 0 */  
#endif  
/* note parentheses around expansion needed in _most_ possible uses  
because ternary is roughly third-lowest precedence,  
and should always be provided to allow for _all_ legal uses */  
/* but DO leave space between name and ( otherwise you are  
instead trying to define a FUNCTION-like macro you don't want */  
....  
#define NO_REPLACE 0  
y = MY_SYMBOL; /* gets 1 or 0 depending on COMPILE_TIME_FLAG */  
#undef NO_REPLACE  
#define NO_REPLACE 1  
y = MY_SYMBOL; /* gets 0 */  
#undef NO_REPLACE
```

Note that `NO_REPLACE` must have some scalar value, preferably 1 or 0, wherever `MY_SYMBOL` is used. You might prefer to organize this as:

```
.... definition as above  
#define NO_REPLACE 0 /* 'normal' case */  
....  
y = MY_SYMBOL; /* gets depending value */  
....  
#undef NO_REPLACE
```

Re: Macro substitution conundrum

```
#define NO_REPLACE 1
y = MY_SYMBOL; /* gets 0 */
#undef NO_REPLACE
#define NO_REPLACE 0
....
y = MY_SYMBOL; /* depending value again */
```

(The relevant subsequence seen by the preprocessor is the same, just the positioning in your source files is different and perhaps easier to keep straight if the NO_REPLACE 1 case is rare.)

This works, but is disgustingly ugly, confusing, and fragile. If your design requires you to do this, better to change your design.

If your NO_REPLACE 1 case is (or can easily be) syntactically nested e.g. only within a function(s), you might even do something like:

```
#define MY_SYMBOL (no_replace_var? 1: otherwise)
extern const int no_replace_var = 0; /* global setting */
....
MY_SYMBOL /* expands to code that should be optimized to
depending value --- but not a constant expression; can't safely
be used in case labels, array bounds (except C99 auto) etc. */
....
int func () {
static const int no_replace_var = 1; /* for this function only */
MY_SYMBOL /* expands to code that should optimize to 0 */
}
```

This is slightly less ugly and may be justifiable in a few rare cases, but only with VERY clear comments and documentation.

– formerly david.thompson1 || achar(64) || worldnet.att.net