How can I conditionally compile based on a preprocessor macro value, while ensuring that the macro is correctly spelled?

devblogs.microsoft.com/oldnewthing/20180910-00

September 10, 2018

Raymond Chen

Continuing on the extremely sporadic topic of stupid C preprocessor tricks.

It is common to have preprocessor macros that control compile-time behavior. C++11 has `std::enable_if` to conditionally remove functions and template specializations from consideration, and C++17 adds `if constexpr` to allow statements to removed conditionally. Removing variables is a bit trickier, though. You can probably manage it by wrapping the variable inside a class that itself uses `std::enable_if`, but that's even more clunky than `std::enable_if` already is.

Anyway, for whatever reason, you might want to use the preprocessor's `#if` directive to perform your tests. Maybe you are preprocessing something for a purpose other than compilation by a C or C++ compiler.

But you're also worried that somebody might misspell your symbol.

```
// The FEATURE_BLAH macro is defined either as 0 or 1

#if FEATURE_BLUH
... do stuff with feature Blah ...
#endif
```

Oops, they misspelled `FEATURE_BLAH`, but the preprocessor doesn't know that, so it happily says, "Nope, it's not defined, skip the body of the `#if`."

How do you catch this typo?

You can use your adversary's power against him.

Since undefined symbols are treated as having the value zero, you can use an expression that blows up if the value is zero.

```
// The FEATURE_BLAH macro is defined either as 1 (off) or 2 (on)

#define GET_NONZERO_VALUE(x) (0/(x) + (x))

#if GET_NONZERO_VALUE(FEATURE_BLAH) == 2
... do stuff with feature Blah ...
#endif
```

The `GET_NONZERO_VALUE` macro first tries to divide by its parameter. If the parameter is not defined or is defined with the value zero, then that results in a division by zero and you get a compiler error. If the parameter is defined with a nonzero value, then the result of `0/(x)` is zero, and adding that to `x` yields `x`.

The last wrinkle is using the `defined` preprocessor pseudo-function to distinguish between an undefined macro and a defined macro whose value is zero.

```
// The FEATURE_BLAH macro is defined to 0 or 1
// The FEATURE_BLAH_OPTION macro is some value

#define GET_FEATURE_VALUE(x) (0/defined(FEATURE_##x) + (FEATURE_##x))

#if GET_FEATURE_VALUE(BLAH)
#if GET_FEATURE_VALUE(BLAH_OPTION) == 1
... do stuff with feature Blah and option 1...
#elif GET_FEATURE_VALUE(BLAH_OPTION) == 2
... do stuff with feature Blah and option 2...
#else
#error Unknown option for FEATURE_BLAH_OPTION.
#endif
#endif
```

If `FEATURE_BLAH` is not defined, then the `defined(FEATURE_BLAH)` will evaluate to zero, and then you get a divide by zero error in the preprocessor. If it is defined, then `defined(FEATURE_BLAH)` evaluates to 1, and the expression `0/1 + FEATURE_BLAH` reduces to just `FEATURE_BLAH`.

This is an abuse of the preprocessor, but it may come in handy in a pinch.

Raymond Chen

Follow