Alternatives to using the #error directive to check whether the compiler even sees you

devblogs.microsoft.com/oldnewthing/20090529-00

May 29, 2009

Raymond Chen

In response to my description of how you can use the `#error` directive to check whether the compiler even sees you, some commenters proposed alternatives. I never claimed that my technique was the only one, just that it was another option available to you. Here are some other options.

scott suggested merely typing `asdasdasd` into the header file and seeing if you get an error. This usually works, but it can be problematic if the code does not already compile. And *of course* it doesn't compile, because the reason why you're doing this investigation in the first place is that *you can't get your code to compile and you're trying to figure out why*. Consequently, it's not always clear whether any particular error was due to your `asdasdasd` or due to the fact that, well, your code doesn't compile. For example after adding your `asdadsads` to line 41 of file `problem.h`, you get the error `Error: Semicolon expected at line 412 of file unrelated.h`. Was that caused by your `asdasdad`? Doesn't seem that way, but it actually was, because the preprocessed output looked like this:

```
asdasdasd
int GlobalVariable;
```

After your `asdasdasd`, all that was generated were a bunch of `#define`s, `#if`s, `#endif`s, and `#include`s. None of them generate output, so the compiler proper doesn't see anything; the preprocessor ate it all. Finally, at `unrelated.h` line 412, a header file finally tried to do something other than just define a macro, and it's only then that the error is detected.

But if you can pick the new error out of the error spew, then go for it. (There are also obscure cases where an extra `asdasdasd` doesn't introduce a new error.)

Since the string `#error` is shorter than `asdasdasd`, and it works in more places, I just go with `#error`.

Another suggestion came from Miguel Duarte who suggested generating the preprocessed file and studying it. That helps, but the preprocessor output tends to be huge, and, as I noted in the base article, `#define` directives don't show up, so it can be hard for you to find your place. I also noted in the base article that if you use Visual Studio's precompiled header files, the contents of the preprocessed output may not match what the compiler sees. In fact, that's the most common reason I've found for a line being ignored: You put the `#include` directive in a place that the preprocessor sees but which the compiler doesn't see because you violated one of the precompiled header consistency rules, usually the source file consistency rule.

Raymond Chen

Follow