# Const methods don’t prevent a method from having side effects

devblogs.microsoft.com/oldnewthing/20180711-00

July 11, 2018

Raymond Chen

In my musing on whether people write insane code with multiple overlapping side effects with a straight face, I noted that raising a warning on any code that depends on the order of evaluation would generate a lot of false positives, such as

```
total_cost = p->base_price + p->calculate_tax();
```

It has been argued that this is merely evidence that the `calculate_tax` method should be const.

Well, except it may not be const. For example, `calculate_tax` needs to look up the tax rate, which means it needs to look up the tax region, and it may decide to cache that information in the object so that future tax calculations can be more efficient. Which means non-const.

And then there’s this:

```
total_cost = apply_discount(p->base_price) + p->calculate_tax();
```

This is still a potential dependency upon the order of evaluation, because the `apply_discount` function might modify the thing that `p` points to.

It’s unlikely, but technically legal.

In practice, nearly everything you write is potentially dependent upon the order of evaluation, but in practice it isn’t because you are not a nincompoop. But the compiler doesn’t know that. The compiler must adhere to the letter of the language standard, because it has to compile insane code as well as sane code.

Maybe you’ll say, “Fine, the compiler shouldn’t complain about potential order of evaluation dependencies in sane code.” But now the argument is circular: What is sane code? Code that isn’t dependent upon order of evaluation.

**Bonus reading**: You don't know const and mutable. Which introduces yet another wrinkle into the story.

Raymond Chen

**Follow**