Do people write insane code with multiple overlapping side effects with a straight face?

devblogs.microsoft.com/oldnewthing/20170719-00

July 19, 2017

Raymond Chen

On an internal C# discussion list, a topic that comes up every so often is asking about the correct interpretation of statements like

```
a -= a *= a;
p[x++] = ++x;
```

I asked,

> Who writes code like that with a straight face? It's one thing to write it because you're trying to win the IOCCC or you're writing a puzzle, but in both cases, you know that you're doing something bizarre. Are there people who write `a -= a *= a` and `p[x++] = ++x;` and think, "Gosh, I'm writing really good code?"

Eric Lippert replied "Yes, there are most certainly such people." He gave as one example a book from an apparently-successful author (sales of over four million and counting) who firmly believed that the terser your code, the faster it ran. The author crammed multiple side effects into a single expression, used ternary operators like they were going out of style, and generally believed that run time was proportional to the number of semicolons executed, and every variable killed a puppy.

Sure, with enough effort, you could do enough flow analysis to have the compiler emit a warning like "The result of this operation may vary depending upon the order of evaluation", but then you have to deal with other problems.

First of all, there will be a lot of false positives. For example, you might write

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

This would raise the warning because the compiler observes that the `calculate_tax` method is not `const`, so it is worried that executing the method may modify the `base_price`, in which case it matters whether you add the tax to the original base price or

the updated one. Now, you may know (by using knowledge not available to the compiler) that the `calculate_tax` method updates the tax locale for the object, but does not update the base price, so you know that this is a false alarm.

The problem is that there are going to be an awful lot of these false alarms, and people are just going to disable the warning.

Okay, so you dial things back and warn only for more blatant cases, where a variable is modified and evaluated within the same expression. "Warning: Expression relies on the order of evaluation."

Super-Confident Joe Expert programmer knows that his code is awesome and the compiler is just being a wuss. "Well, *obviously* the variable is incremented first, and then it is used to calculate the array index, and then the result of the array lookup is stored back to the variable. There's no order of evaluation conflict here. *Stupid compiler*." Super-Confident Joe Expert turns off the warning. But then again, Super-Confident Joe Expert is probably a lost cause, so maybe we don't worry about him.

Joe Beginner programmer doesn't really understand the warning. "Well, let's see. I compiled this function five times, and I got the same result each time. The result looks reliable to me. Looks like a spurious warning." The people who would benefit from the warning don't have the necessary background to understand it.

Sure enough, some time later, it came up again. Somebody asked why `x ^= y ^= x ^= y` doesn't work in C#, even though it works in C++. More proof that people write code that rely upon multiple side effects, and they passionately believe that what they are doing is obvious and guaranteed.

Raymond Chen

Follow