

The Old New Thing

What is the order of evaluation in C#?

14 Aug 2007 10:00 AM

68

The C and C++ languages leave the order of evaluation generally unspecified aside from specific locations called *sequence points*. Side effects of operations performed prior to the sequence point are guaranteed visible to operations performed after it.¹ For example, the C comma operator introduces a sequence point. When you write `f(), g()`, the language guarantees that any changes to program state made by the function `f` can be seen by the function `g`; `f` executes before `g`. On the other hand, the multiplication operator does not introduce a sequence point. If you write `f() * g()` there is no guarantee which side will be evaluated first.

(Note that order of evaluation is not the same as associativity and operator precedence. Given the expression `f() + g() * h()`, operator precedence says that it should be evaluated as if it were written `f() + (g() * h())`, but that doesn't say what order the three functions will be evaluated. It merely describes how the results of the three functions will be combined.)

In the C# language, the order of evaluation is spelled out more explicitly. The order of evaluation for operators is left to right. if you write `f() + g()` in C#, the language guarantees that `f()` will be evaluated first. The example in the linked-to page is even clearer. The expression `F(i) + G(i++) * H(i)` is evaluated as if it were written like this:

The side effects of each part of the expression take effect in left-to-right order. Even the order of evaluation of function arguments is strictly left-to-right.

Note that the compiler has permission to evaluate the operands in a different order if it can prove that the alternate order of evaluation has the same effect as the original one (in the absence of asynchronous exceptions).

Why does C# take a much more restrictive view of the order of evaluation? I don't know, but I can guess.²

My guess is that the language designers wanted to reduce the frequency of a category of subtle bugs (in this case, order-of-evaluation dependency). There are many other examples of this in the language design. Consider:

The language designers specified that the scope of a local variable in C# extends to the *entire* block in which it is declared. As a first consequence of this, the second declaration of `i` in the function `f()` is illegal since its scope overlaps with the scope of the first declaration. This removes a class of bugs that can be traced to one local variable masking another with the same name.

In the function `g()` the assignment `x = 3;` is illegal because the `x` refers not to the member variable but to the local variable declared below it. Notice that the scope of the local variable begins with the entire block, and *not* with the point of declaration as it would have been in C++.

Nitpicker's Corner

¹This is a simplified definition of *sequence point*. For more precise definitions, consult the relevant standards documents.

²I have not historically included the sentence "I don't know but I can guess" because this is a blog, not formal documentation. Everything is my opinion, recollection, or interpretation. But it seems that people take what I say to establish the official Microsoft position on things, so now I have to go back and add explicit disclaimers.

Blog - Comment List MSDN TechNet

Comments



Karellaen #

14 Aug 2007 10:18 AM

Uh - hang on. If shadowing variables is now allowed and `i` cannot shadow `i`, how come `x` can shadow `x`? The outer `x` is declared in the block defining "class A", so should overlap the `x` in `A::g`.

Shouldn't it? Or have I misread something?

Second, if the scope of a variable covers the whole block, not just from the point of declaration, why can't you do:

```
class A {
    void f() {
        x = 3;
        int x;
    }
}
```

Doesn't the fact that that doesn't work mean that the variable's scope *does* begin only at the point of declaration? If not, what does saying that a variable's scope begins at the start of the block mean? How is it different from its scope starting at the point of declaration?

confused

[Don't be lazy. I'm not going to do your homework for you. Read the C# language specification. Section 5.1.7 ("Local Variables") will be useful. -Raymond]