The scope of the C# checked/unchecked keyword is static, not dynamic

devblogs.microsoft.com/oldnewthing/20140815-00

August 15, 2014

Raymond Chen

C# has operators `checked` and `unchecked` to control the behavior of the language in the face of integer overflow. There are also `checked` and `unchecked` statements which apply the behavior to blocks of statements rather than single expressions.

```
int x;
x = checked(a + b); // evaluate with overflow checking
y = unchecked(a + b); // evaluate without overflow checking
checked {
 x = a + b; // evaluate with overflow checking
}
unchecked {
 x = a + b; // evaluate without overflow checking
}
```

Why, then, doesn't this code below raise an overflow exception?

```
class Program {
 static int Multiply(int a, int b) { return a * b; }
 static int Overflow() { return Multiply(int.MaxValue, 2); }
 public static void Main() {
  System.Console.WriteLine(checked(Overflow()));
  checked {
    System.Console.WriteLine(Overflow());
  }
 }
}
```

(Mini-exercise: Why couldn't I have just written `static int Overflow() { return int.MaxValue * 2; }` ?)

The answer is that the scope of the `checked` or `unchecked` keyword is static, not dynamic. Whether a particular arithmetic is checked or unchecked is determined at compile time, not at run time. Since the multiplication in the `Multiply` function is not explicitly marked checked or unchecked, uses the overflow context implied by your compiler options. Assuming

you've left it at the default of `unchecked`, this means that there is no overflow checking in the `Multiply` function, even if you call it from a checked context. Because once you call the `Multiply` function, you have left the checked context.

The C# language specification addresses this issue not once, not twice, but three times! (But it seems that some people miss it, possibly because there is too much documentation.)

First, there is an explicit list of operations which are controlled by the `checked` or `unchecked` keyword:

> - The predefined `++` and `--` unary operators, when the operand is of an integral type.
> - The predefined `-` unary operator, when the operand is of an integral type.
> - The predefined `+`, `-`, `*`, and `/` binary operators, when both operands are of integral types.
> - Explicit numeric conversions from one integral type to another integral type, or from `float` or `double` to an integral type.

That's all. Note that function calls are not on the list.

Now, that may have been a bit too subtle (documentation by omission), so the language specific goes ahead and calls it out.

> The `checked` and `unchecked` operators only affect the overflow checking context for those operations that are textually contained within the "`(`" and "`)`" tokens. The operators have no effect on function members that are invoked as a result of evaluating the contained expression.

And then, in case you still didn't get it, the language specification even includes an example:

> ```
> class Test
> {
>    static int Multiply(int x, int y) {
>       return x * y;
>    }
>    static int F() {
>       return checked(Multiply(1000000, 1000000));
>    }
> }
> ```
>
> The use of `checked` in `F` does not affect the evaluation of `x * y` in `Multiply`, so `x * y` is evaluated in the default overflow checking context.

(I wrote my example before consulting the language specification. That we both chose to use multiplication overflow is just a coincidence.)

Even though the language specification says it three times, in three different ways, there are still people who are under the mistaken impression that the scope of the `checked` keyword is dynamic.

Another thing you may have notice is that the `checked` and `unchecked` keywords apply only to the built-in arithmetic operations on integers. They do not apply to overloaded operators or to operators on custom classes.

Which makes sense if you think about it, because in order to define an overloaded operator or an operator on a custom class, you need to write the implementation as a separate function, in which case you have already left the scope of the `checked` and `unchecked` keywords.

And now we are leaving the scope of CLR Week. You can remove your hands from your ears now.

Raymond Chen

Follow