

## Re: Setting pointer to null!

---

*Source:* <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2009-04/msg00063.html>

---

- *From:* "Doug Harrison [MVP]" <dsh@xxxxxxxx>
  - *Date:* Fri, 03 Apr 2009 14:21:12 -0500
- 

On Fri, 3 Apr 2009 16:47:26 +0300, "Alex Blekhman" <tkfx.REMOVE@xxxxxxxx> wrote:

"Doug Harrison [MVP]" wrote:

I'm afraid you'll have to explain to me what you're "differing" with, because it seems to me you've restated what I said.

I disagree with the following statement:

"The best thing is to assign a non-NULL singular value." And consequently: "In any case, if you "defensively" initialize all pointers to NULL, you defeat [filling variables with predefined value] debugging feature!".

Let me explain. I think this debugging feature emerged originally in order to mitigate the uninitialized variables problem. When developer forgets to initialize a variable and it accidentally gets zero value at the run time, then this omission may exist unnoticed for a long time. That's why the debugger cares to fill uninitialized vars with deliberate garbage, so developer will notice the omission on the first attempt to use the variable.

This behavior of the debugger based on the premise that variables should contain valid data for their lifetime.

Yes, the purpose of this feature is to detect the use of uninitialized variables, and I thought that was understood. Remember that in C, local variables could only be declared at the tops of blocks, and it was common to declare a bunch that would be "initialized" in relatively distant code. In C++, this is much less of an issue, because you can declare variables anywhere, and the space over which they exist but really should not exist is greatly reduced.

In any case, if you always initialize pointers to NULL, including ones for which NULL is not a valid value at that point in time, you defeat the

## Re: Setting pointer to null!

debugging feature, because you prevent the compiler from initializing them with the special pattern. (You also run the risk of introducing new logic errors, as I've described a couple of times already.) That's what I've been saying, and I don't see how it's disputable, so I still don't understand what you're disagreeing with. It really puzzles me. The only thing I see that's not perfectly clear you haven't commented on, and that was my statement about assigning "a non-NULL singular value". Given that I went on to talk about the compiler doing this (which you clipped and replaced with your own summary – please stop doing that) I was really thinking of what the compiler does, and obviously if you were to do it yourself, you would need to use the same value the compiler uses.

Valid data includes zero value, which indicates that the variable is empty or whatever. The same as the class invariant notion in the C++.

But 0 is not a valid value for every variable at every point in time. Just because some value is in the domain of a type does not imply it's valid for every variable of that type. It may impart a "predictable" behavior, but it doesn't mean it's valid, and invalid data can cause incorrect behavior, as I've already described a couple of times.

Indeed, the following initialization of the `p` pointer seems redundant:

```
T* p = 0;
if (...)
{
... not using p
p = ...
}
else
{
... not using p
p = ...
}
```

However, we all know too well that such code quickly becomes

```
T* p = 0;

// do something else

if (...)
{
... not using p
p = ...
}
```

## Re: Setting pointer to null!

```
else
{
... not using p
p = ...
}
```

Where "do something else" part may involve an operation on the `p` pointer. Once written even the excellent code becomes mediocre as the time goes. Eventually mediocre code becomes bad code. This kind of defensive programming will save the future developer unnecessary aggravation.

I can't agree that's even remotely common, simply because you don't just start using a variable without knowing anything about it. Or if you do, you quickly learn what a mistake it is, and you stop. In C++, it especially does not happen, because there's no reason to extend the lifetime of `p` in that way.

That's why it is often said that initializing a variable is a good coding practice. It ensures that a variable contains valid data throughout its lifetime.

The real goal is to minimize the distance between declaration and the assumption of a valid value. IOW, it's all about lifetime control, because if a variable does not exist, it doesn't matter what value it would or wouldn't have. Of course it's best to establish the initial (valid!) value with initialization; I certainly have not been disputing that.

Unlike languages like C# and Java, C++ does not have "definite assignment". It does, however, allow you to declare variables near their point of first use, which mitigates the issue quite a bit for those variables that can't be initialized when they are declared. Look at the examples here:

[http://java.sun.com/docs/books/jls/second\\_edition/html/defAssign.doc.html](http://java.sun.com/docs/books/jls/second_edition/html/defAssign.doc.html)

I can't believe you would insist upon initializing every "k" in these examples. And please note that both C# and Java have "definite assignment", \*not\* "required initialization".

--

Doug Harrison  
Visual C++ MVP

.