

## Re: faster access private or public?

**Source:** <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2004-09/1398.html>

---

**From:** Igor Tandetnik ([itandetnik\\_at\\_mvps.org](mailto:itandetnik_at_mvps.org))

**Date:** 09/28/04

Date: Tue, 28 Sep 2004 10:46:19 -0400

"Sigurd Stenersen" <[sigurds@utvikling.com](mailto:sigurds@utvikling.com)> wrote in message  
news:OcBuZ6UpEHA.744@TK2MSFTNGP10.phx.gbl

> *I guess I have to spell it out...*

>

> `void g(int i)`

> {

> `OhFooey->~Foo();`

> `new (OhFooey) Foo(10);`

> }

>

> *In this case it \*is not\* okay for the compiler to assume that private  
> variables are not changed when optimizing some other piece of code  
> that uses the instance pointed to by OhFooey and that code calls g().  
> I believe we all agree on that ?*

Right. But this code has well-defined behavior, so I don't quite see how  
it is relevant to the question at hand.

> *Now, what you're saying seems to be that in this case...*

> `void g(int i)`

> {

> `new (OhFooey) Foo(10);`

> }

> *...it \*is\* okay for the compiler to make that same assumption.*

Only if the compiler can somehow prove that this code is always called  
in a context where it causes undefined behavior. It would take a very  
smart compiler to do that, though. You cannot say that just looking at  
the code of `g()` in isolation, since it can be used legally, as in:

```
OhFooey->~Foo();
```

```
g(1);
```

> *I don't understand why you think this is OK, but anyway could you  
> tell me how the compiler is going to know what is actually being done  
> in this part of the code while optimizing some other piece of code ?*

microsoft.public.vc.language: Re: faster access private or public?

It does not matter what the compiler knows or does not know, or what reasoning it employs or does not employ. The moment a code that exhibits undefined behavior is executed, you cannot argue any longer whether the compiler-generated code is right or wrong. By definition of undefined behavior, anything the program might do from that point on is correct. You cannot argue that the code is wrong, because to argue that, you have to say: I expected the code to behave in such and such a way, but I observe it behaving differently, therefore it must be wrong. By coding undefined behavior, you lose the right to claim any expectations as to the compiler output, so you cannot clear the first hurdle of that two-pronged test (yes, I had to read a lot of legal documents lately).

--

With best wishes,

Igor Tandetnik

"On two occasions, I have been asked [by members of Parliament], 'Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?' I am not able to rightly apprehend the kind of confusion of ideas that could provoke such a question." -- Charles Babbage