

Re: calloc.... Why?

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.cpp/2004-07/2814.html

From: tom_usenet (tom_usenet_at_hotmail.com)

Date: 07/20/04

Date: Tue, 20 Jul 2004 14:38:53 +0100

On Tue, 20 Jul 2004 12:30:03 GMT, JKop <NULL@NULL.NULL> wrote:

>tom_usenet posted:

>

>> *And using references doesn't make the optimizer's job any easier IMHO.*

>> *But it's likely that both versions will generate identical code when*

>> *inlined in the same context.*

>>

>> *Tom*

>

>*It's not optimization at all, it's simply what inline functions are all about. When a function is "outline", arguments have to be passed to it in a certain way (stack and registers), but when the function's inline, the stack and registers aren't bothered with.*

inline is only a hint to the compiler, and a way to avoid violating the one definition rule. The compiler can (and does) choose to inline apparently non-inline functions, and to put inline functions out of line.

So... with an inline

>*function; if you use const references, the compiler has no optimization to do, it simply uses the objects in the calling function.*

But it introduces aliasing problems, making optimization harder, and can even change semantics. I'll give you an example:

```
int foo = 10;
```

```
inline int f(int const& ref)
{
    foo = ref;
    ++foo;
    return foo + ref;
}
```

```
}  
  
int main(int argc, char** argv)  
{  
    int const& i = argc > 1? foo: 10;  
    f(foo); //hmmm.  
}
```

Because a reference is used, the compiler doesn't know whether that reference is an alias for "foo", so it has to assume it is, and therefore produce suboptimal code. If you change the semantics to:

```
inline int f(int ref)  
{  
    foo = ref;  
    ++foo;  
    return foo + ref;  
}
```

then the compiler can optimize it much better, since it knows "ref" is free from aliases (since it is a local variable).

But my basic point is that references can be bad since they are aliases, and aliases hurt optimization, since you never know what they are aliasing.

On another note, your concept of "use the objects in the calling function" doesn't really make sense. Remember that CPUs have registers, and often variables have their values both in "main" memory and in a register. When inlining a function, whether using by-ref or by-val parameter passing, the compiler optimizes things to work out whether it already has the relevant value in registers, or whether they have to be reloaded, or whatever. Using references vs values may actually harm this process, since the optimizer is generally happier dealing with local, independent variables.

Still, this is a complex issue, and examining output assemblies is probably the best way to experiment.

But if the arguments are by-value, then the
>compiler actually would have to do an optimization to
>realize that it can use the objects from the calling
>function. Alternatively you could pass by-value and have
>the objects const in the inline function, then the compiler
>would know to use the objects from the calling function. I
>prefer the const references, as it's very clear what's
>going on.

However, it goes against all style guidelines I've read, and will therefore confuse anyone reading your code (as you have already seen with this thread).

Re: calloc.... Why?

comp.lang.c++. Re: calloc.... Why?

Tom