Subject: Re: C++ FQA
Posted by mdelfede on Fri, 09 Nov 2007 12:41:56 GMT
View Forum Message <> Reply to Message

luzr wrote on Fri, 09 November 2007 08:51
Well, but that is useful feature and this is one of things I like with C++ - the "default" mode is "safe", but you can always do dirty things when you need them.

no, here I don't agree.... such things make virtually impossible write 'safe' libraries. A 'const' should be a 'const', not a 'maybe const'.... as 'private' should be so, not a maybe one.
I've seen constructs like that :

#define private public
#include "alib.h"

just to overcome a private class declaration and access the low-level stuffs inside it.... Then, when library changes, people (maybe also people that hasn't nothing to do with such a hack) starts wondering why his program that up to the day before worked like a charm just crash. IMHO that has nothing to do with commercial-grade applications.

Quote:
In other words, you can also say that a well written library can do what it needs

well, a well written lib should do what the coder will, *not* what the user is missing. Before using C++ hacks to overcome libs limitations, you have 3 solutions :
1) Patch the sources, if you have them
2) Ask the original programmer to enhance the lib
3) Just find another lib that suit your needs

Quote:
Actually, interestingly it seems like I am the only one here who in fact likes C++ as it is (except some quite small issues and the standard library, which IMO only looks like a good design).

Well, I agree that C++ *is* useful and *is* the only widespread system-wide programming language. But I really can't say that is a good language. Besides static memory management, which I prefere against a gc approach (I like do code what I want, not what the compiler want...), it contains really too many caveats dues mostly (but not all) because of compatibility issues.
It is :
- slow compiling
- not modular at all
- object model is missing too many useful stuffs (properties, delegates, a true rtti system, just among all)
- operator overloading is just awful, as is missing rvalue-lvalue different behaviour
- missing high-level types (strings, arrays.....)
- cumbersome templates
- no binary objects specifications... in particular with respect to to name mangling
- this damn'd preprocessor that does what he wants

Just an example about this... on a really poorly written code :

#define a_type mytype
#define an_include </my/include/dir/a_type/mytype.hxx>
#define another_include "/my/include/dir/a_type/mytype.hxx"
#include an_include
#include another_include

That has the wonderful (sigh) result of :

#include </my/include/dir/mytype/mytype.hxx>
#include "/my/include/dir/a_type/mytype.hxx"

I stumbled about such a problem and it tooks half a day to understand that inside <> you have macro substitution, but inside "" not.... and I'm still not sure that it isn't a compiler behaviour.

IMHO, what we needs is a new system wide language, that maybe resembles to C++, but gets rid of all caveats and introduces the missing things. C++ is a language that, in order to be able to compile 1980's code is just becoming a monster and still missing what a modern oo language should have.

Ciao

Max