Subject: Ultimate++ possible adaptation to C++0x
Posted by ptDev on Sun, 14 Jun 2009 10:45:19 GMT
View Forum Message <> Reply to Message

Hi everyone,

I believe it's been discussed somewhere else that the next C++ standard, although solving issues in the STL, will not do that much for U++.

Even being aware of this, I would like to ask the project maintainers the following:

- Will you use the upcoming move semantics (&&) to make the NTL more flexible in this regard? I would like to more easily be able to copy by assignment, rather than move, when I choose to.

- Would you consider applying template concepts to the Ultimate++ library design? If you did this, what would your approach be?

I'm very curious to see how U++ evolves along with the language, seeing that other GUI libraries were hesitant to even introduce templates in this day and age...

Subject: Re: Ultimate++ possible adaptation to C++0x
Posted by cbpporter on Sun, 14 Jun 2009 16:21:43 GMT
View Forum Message <> Reply to Message

Well my contribution are more related to reporting and fixing bugs and adapting U++ for my own not-general needs rather than doing huge stuff like adapting NTL to C++0x, but still I'll give my opinion.

I think that C++0x is generally a great addition to C++ which will hugely benefit STL users, but won't really have any noticeable effect on U++.

First, the issue of move semantics. In C++0x move semantics will allow for transparent move semantics and to optimize performance wise the use of assignment operator, which will be used both for copy and move. Assignment operator plays both roles.

In U++, we have assignment play the role of move in most NTL classes, but sometimes it is copy. But the general rule is that it is a move. But we do have an easy way of doing a copy in "<<=" operator. So if in U++ "=" is move and "<<=" is copy and in C++0x "=" is both, I think we have a problem here. Taking the C++0x approach would certainly be problematic in the initial phase and would invalidate also our old habits. Obtaining the ability to write complex code which uses a mix of move and copy is not something you get over night, and this stands for both STL type of programming and for NTL. I think that things are going to remain the same for quite a while after C++0x is finally published as a final standard and first compilers have appeared. If it were up to me, I wouldn't change anything, but Mirek might choose so as time passes. Maybe adding new "default" operators which take r-value parameters.

As for other features of C++0x:

- constexpr: not bad, but I have never had problems with the relatively underpowered constness system from C++. The fact that I'll be able to write compile time functions which evaluate to a constant won't alter the way I write C++ and is probably not something that will come up that often.
- external templates: hell yeah. If they get this right and you can concentrate all common instantiations to a single compilation unit, this could theoretically reduce compile time in a noticeable way and produce small object files. I tend to avoid excessive templates and inlining at interface level. I am yet to install GCC 4.4 and see if this allegedly already implemented feature lives up to it's potential.
- initializer lists: maybe we'll get appropriate constructors, but adding items to NTL containers is fairly easy right now.
- uniform initialization: ones it gets out in enough compilers I'm sure to use it as much as possible, but I doubt anybody will go over U++ code and start using it. Maybe for new code.
- auto: again, a blessing for STL and Boost, where you may have complex types and iterators, but in U++ we almost always use very simple classes were I have no problem determining at a glance what type is in question. Dubious use for us.
- range based loop: good, but again not as great for us because out loops are index based.
- lambdas: I'm not touching that ugly syntax with a ten foot pole
- concepts: we should add transparently support for concept checking for nice and clear error messages to our templates.
- variadic templates: we have quite a number of templates where there are different versions based on number of arguments. Maybe we could replace them with this.
- new string literals: I'm hoping this will have a positive impact on everything. Most IT products are still in Unicode stone age, and everything from standard libraries is well before stone age, probably primordial soup stage.
- user defined literals: cute, but I won't clutter my already complex C++ code with this.
- other features are small but generally useful. I don't see them changing the way C++ is written.

These are of course only my personal opinions. Mirek stated that he'll ignore C++0x for now, and I agree, except for those external templates.
And sadly for me, almost everything that I hoped for in the new standard first was not included.

My biggest complain: first of all leave the language alone and fix it's most blatant flaw: the lack of a proper module support and text based inclusion. We all saw that Java, C#, D and ObjectPascal (and even plain old Pascal) just to name a few have extremely short compile times because the compiler is not reading, parsing and compiling the same huge text files over and over in each compilation unit. These languages do that exactly once, and the results are available for future operations. It is possible to save the interface description of a module in a way optimized for reading and linking.

My second complain: C++ is not something you add things to. C++ is so huge, that it would have benefited form a huge cut in duplicate features, even if it would have meant the loss of compatibility. Even so, mixing compilation unit from old and new C++ would have been possible to a certain extent, the way one can call C functions, and in 10 years maybe people would have thanked the ones responsible for the standard for their bold move. Or maybe they would have brought out the forks and torches (and guns). But with the new standard, C++ is so bloated that I think no one can because a 100% expert in C++ without an unpractical time investment. I would like a language where young graduates have good solid knowledge in a language that incourages

this, not a situation where they have limited knowledge of C++ and the potential that after 10 years they will be the next top-class experts in their language. Not considering the sizable library, in the case of Java, take a smart student with great knowledge in algorithmics, teach him Java and good coding practices for a year and all that individual needs is some practice on real live projects and he is well on his way of becoming a competent Java expert in an ideal world.

Subject: Re: Ultimate++ possible adaptation to C++0x
Posted by mirek on Sun, 14 Jun 2009 21:43:22 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Sun, 14 June 2009 12:21
First, the issue of move semantics.

The real showstopper of move semantics is that it lacks automated composition.

That would require quite a lot of additional code in some places where we are using picks...

(But time will tell. Maybe these places are in fact not so numerous to justify _pick..)

Mirek

Subject: Re: Ultimate++ possible adaptation to C++0x
Posted by cbpporter on Mon, 15 Jun 2009 20:59:18 GMT
View Forum Message <> Reply to Message

I just finished testing GCC 4.4. It compiles everything OK and is insignificantly slower probably because of insignificant increase in object file size.

Then I tested it's C++0x support. I had to change a single line where G++ complained, but otherwise U++ compiles in C++0x mode.

As for testing external templates, they work. But I see no change: the resulting object files are just as big (in one test even bigger by a few kb) and compilation time is the same. So what are then these external templates supposed to help with? I see no reason to use them in GCC 4.4, since there is a good chance that some inline template functions will not be inlined and there was zero gain for me.
Or maybe I need a better test case.

Maybe in future versions it will work better, but I got a strange feeling that the feature I've been looking most forward in new standard will ultimately disappoint me. I guess I was expecting something that would work as well as D templates and I was barking up the wrong tree.
