Subject: Re: Pick overloaded by Rvalue?
Posted by cbpporter on Sat, 22 Nov 2008 09:04:36 GMT
View Forum Message <> Reply to Message

captainc wrote on Fri, 21 November 2008 20:48Yeah, I saw the date and was amazed that it would take till then. I thought they were not going to get features in because the release date was earlier. I don't know how I feel about that late date. I was excited to see C++ take its next step sooner.

In all, this just validates the need right now for a library like U++; Especially with its alternative approaches in the core libraries.

Though, some thought should be put in to how U++ will change when the new standard comes out.
Yes, this is not how the situation was presented to the public. After jocking for years "C++0x, where x will be decimal we hope, not hexadecimal", and with the added requirement of submiting the draft in 2008 so it can be acepted by 2009, you would think that since they are on schedule, we would have the draft in 2009. But no, after 2009 there are going two be two years to work out the bugs. So we will get the final standard pretty late.

And until we get compiler support much time will pass. Good news is that we will probably get to play around with some features sooner. By 2009 the quite limited list of C++0x features supported by GCC should have an addition or two. And by 2010, 2011 we should have some support for the uncorrected draft. Question is how portable is that going to be. I think MSC will not experiment the same way GCC does and will try to bring out a mostly complete C++0x compiler somewhere in the future, which personally I hope it doesn't. I sure love MSC, not because it is from MS or anything,just because it's good and fast (and IMO a whole lot better than GCC performance wise; it's not just like MSC is 5% faster or some small number like this), so a C++0x MSC compiler would be a good thing. But I would like them to not give attention to anything else except .NET. I'm hoping even for a Windows version which has a .NET with fewer limitations and without WinAPI. On the other hand, GCC is still great for a production tool (especially when you don't have comparison). If they provide a compiler with uniform initialization, auto, concepts and rvalues, I will be sure to use it, even if is not 100% stable, and I'm sure others will use it too.

But I personally consider C++ doomed. And the final stab that killed it is C++0x. Not that C++0x is not good. It is actually so good that I wish it would have been brought out as C++03. But it's not. And with all those new good features plus good features from previous standards, if you don't have compatibility problems with older software, I'm sure you will be able to use C++ for some time now. But with the rest of the more ugly features, C++ is extremely bloated. What is going to happen after C++0x? TR1? Sure. Add modules to C++ and I'll shut up and probably be able to use happily C++ until I retire if I so desire. Add transparent GC and prove to the world that the performance penalty is not worth getting excited over. Add a really rich MT library (C++0x's is not that rich, only standardized)? Great!!! All these great feature, what could go wrong. Well the bloat can go wrong. Even if you use only the new features, you are going to have to have working knowledge of all features and this is where the amount of bloat will show it's ugly face. And TR1 is as far as you can go. You can't really bring out a new standard later, because of the bloat. A new programmer will have to learn C++ for years without end, and if the programmer hopes to reach the level of knowledge of C++ veteran switch have gone through each iteration of the language,

than a time investment equivalent to their's will be necessary. You will probably have problems with hiring a graduate to work on a C++ project which uses a lot of features. That is true Today. I know a lot of programmers, but few can handle C++ and even less desire to. On the other hand, if you use C++ as C with classes, only use references as const & parameters, anybody can handle C++ and bring a project from start to finish. And that is happening a lot around me.

As for how this will affect U++, it is very hard to tell. First let's consider how much will U++ evolve until then. Personally I think it will remain pretty much the same. I hope will have a moment where we clean it up a little, make it even more modular and remove deprecated stuff. I think we should embrace a 3 level/release approach for deprecation. In the current release we find something that must be deprecated, we move it to level one and we keep it there in next release. Once we have reached that release, we move it to level 2, and keep it there for the next release. Level 2 is not activated by default,so you have to add a flag. To give an example, we find deprecated stuff in 2008.1. While working on the between release versions, we mark it as level one. So in 2008.1 and in 2009.1 we still have the deprecated features and can only be removed by an explicit flag if a user desires. This gives two years for the feature: one of pre-releases, dev versions,and one real release. From the moment 2009.1 is out, level1 items move to level 2, which is not activated by default and new deprecated items move to level one. This leaves you in the situation that a dev after a release removes deprecated items, but that's what dev should do: they can break compatibility. And level two items can always be turned on by a simple flag. And with 2010.1, new items come to level 1, level 1 becomes 2, and level 2 becomes level 3. Level3items are removed.

There are alot of variations on this scheme if it seems to complicated.

Also new packages are sure to surface. Also, I except a larger user base. Not large compared with other toolkits, but large enough that we will have troubles resolving all the requests and support on this forum. The problem could be easily solved with a couple of additional Mireks .