Subject: I don't get some aspects of STL ... [pointless rant]
Posted by mr_ped on Wed, 18 Mar 2009 20:45:27 GMT
View Forum Message <> Reply to Message

I'm forced to use STL for some short piece of code, and I'm wondering whether this is really that great of language extension... (also I became so used to NTL naming and pick behavior, that it's a real pain now for me to watch how STL works [inefficiently])

At first I did need to put some strings into hash map, so I found out std::hash_map .. then I figured out it's renamed to something new in 4.3.x and finally part of default stlib, but in older compilers it's just some extension which may be not available (my case). At that point I was so sick of the situation, I simply took my old source with CCIT 16bit CRC which works as excellent 16b hash function for strings with minimal CPU usage, created 64k of vector<int>, add/get functions, and there we go, problem solved.

Then I did want to sort some strings and get the conversion data of old indices to new indices, so I can reorder also additional data in different structures. There's some clever sort variant in U++ doing exactly this (can't recall it's name), but to my surprise there's nothing like that in STL. Although after little bit of thinking I figured out something like this:

```
struct ParentClass::S_compare {
 const ParentClass & p;
 S_compare( const ParentClass & parent ) : p( parent ) {}
 bool operator() ( int i, int j ) {
  return ( /* desired compare on p.get(i) vs p.get(j) */ );
 }
};

void ParentClass::Sort( std::vector<int> & neworder )
{
 neworder.clear();
 for ( int oldid = 0; oldid < /*count of data*/; ++oldid )
  neworder.push_back( oldid );
 S_compare cmp(*this);
 sort( neworder.begin(), neworder.end(), cmp );
 //neworder contains old indices in proper order
}

//call Sort, and use the result in "neworder" to access your data in sorted manner
```

does what I need. A tad more code then simple 2~3 lines in U++, but not that bad, and actually it's quite powerful for extraordinary cases when you need some really crazy compare functions.

Then the third crash into STL today finally made me to write this rant here. I tried to use std::set_difference ... and there's no "in place" variant working on the range1 data as output too, which IMHO makes perfect sense when std::list are used and you don't need original data anymore. And as far as I can tell, there's no problem to write such algorithm (I will try in next minutes, so if there's some problem, I will learn the hard way).

And the vector_type::const_iterator b = v.begin(), e = v.end() is killing me. I'm all about verbose variable names (I rarely use "b" or "v", this is just fake example), but the "vector_type::const_iterator" is overkill for me, it makes me always to think how to not create a "for" loop, or rather use the int i variant, because it's too much writing.

I mean, the STL doesn't look that great to me. I can see how it saves some work in small applications, but I can easily imagine to take back some of that advantage with that verbosity of type names and definitions (see also the compare function embedded into struct). And for big things where performance matters the performance of STL makes it unusable. So far my real life experience with NTL made me always much more happy.

I either didn't understand the "spirit" of STL, or it sucks a bit. If anyone has some good comments to learn me more about it's spirit - so it will be less pain to use it effectively for me, I will be grateful. I will very likely still prefer NTL, but sometimes NTL is not available, while STL is.