Is the CPH STL an active library?

Jyrki Katajainen (University of Copenhagen)

Joint work with Bo Simonsen (University of Copenhagen)

11 May 2009

These slides are available at http://cphstl.dk
Active library

requirements specification

automatic manufacturing

family of components

highly customized and optimized component

Read [Czarnecki et al. 2000]
Generic library like the CPH STL

Visit http://cphstl.dk
Selection of the copying method

**Potential optimization:** If both in the source and the target the elements are stored in a contiguous memory segment, if the elements are POD types, and if the sizes of the elements in both arrays are the same, copy the elements using the fast `memcpy` function, which is available at the standard C library.

**Problem:** The technical report on C++ library extensions does not specify under what circumstances, if any, `std::tr1::is_pod<V>::value` is true.
#include <cstdlib> // defines std::size_t
#include <cstring> // defines std::memcpy
#include <iterator> // defines standard iterator traits
#include <tr1/type_traits> // defines standard type traits
#include "type.h++" // defines cphstl::int2type

namespace cphstl {
  namespace {
    enum copy_algorithms { fast, conservative };

    template <typename I, typename O>
    O copy(I p, I q, O r, cphstl::int2type<fast>) {
      std::size_t n = q - p;
      std::memcpy(r, p, n * sizeof(*p));
      return r + n;
    }

    template <typename I, typename O>
    O copy(I p, I q, O r, cphstl::int2type<conservative>) {
      for (; p != q; ++p, ++r) {
        /* implementation */
      }
    }
  }
}

22     *r = *p;
23 
24     return r;
25 }
26 }
27 }
28 }
29 }
30 }  
31 }  
32 enum {  
33     algorithm =  
34         std::tr1::is_pointer<I>::value &&  
35         std::tr1::is_pointer<O>::value &&  
36         std::tr1::is_pod<V>::value &&  
37         std::tr1::is_pod<U>::value &&  
38         sizeof(V) == sizeof(U) ? fast : conservative  
39     };  
40     return copy(p, q, r, cphstl::int2type<algorithm>());
41 }
Selection of the storage policy

Potential optimization: If it is more expensive to copy an element than a pointer, store elements indirectly; otherwise store them directly.

Problem: No compile-time `costof` operator provided! We can only approximate this optimization.
```cpp
#include <memory> // defines std::allocator
#include "vector-framework.h++" // defines cphstl::vector_framework
#include "dynamic-array.h++" // defines cphstl::dynamic_array
#include "plain-entry.h++" // defines cphstl::plain_entry
#include "safe-entry.h++" // defines cphstl::safe_entry
#include <tr1/type_traits> // defines standard type traits
#include "type.h++" // defines cphstl::if_then_else

namespace cphstl {
    namespace {

        template <typename V, typename A>
        class kernel_selector {
        public:
            typedef cphstl::plain_entry<V, A> E;
            typedef cphstl::dynamic_array<V, A, E> K;
            typedef cphstl::safe_entry<V, A> F;
            typedef cphstl::dynamic_array<V, A, F> L;
            typedef typename cphstl::if_then_else<std::tr1::is_class<V
                >::value, L, K>::type type;
        }
    }
}
```
template <
    typename V,
    typename A = std::allocator<V>,
    typename R = cphstl::vector_framework<V, A, typename ...
        kernel_selector<V, A>::type>,
    ...
> class vector {
    ...
};

Conclusions

- In C++ the facilities provided for compile-time reflection and template metaprogramming are still primitive.
- Yes, we have to take the step from generic programming to generative programming.
Mini-project

create($K$). Create an empty $K$.

destroy($K$). Destroy an empty $K$.

size($K$) const. Return the number of elements in $K$.

capacity($K$) const. Return the current capacity of $K$.

access($K$, $i$) const. Return a reference to the $i$th element of $K$.

access($K$, $i$). Return a reference to the $i$th element of $K$.

grow($K$, $\delta$). Increase the size of $K$ by $\delta \in \mathbb{N}$.

shrink($K$, $\delta$). Decrease the size of $K$ by $\delta \in \mathbb{N}$.

Assignment: Implement a new vector kernel for the CPH STL.