

# The STL Landscape (a partial view)

## Containers

Sequence: `vector, deque, list, basic_string, valarray`  
 Associative: `set, multiset, map, multimap`  
 Derivative: `stack, priority_queue`

## Iterators

```
std::iterator
    difference_type, value_type, pointer, reference, iterator_category
↑ Input Iterator 1
    It it(a) ;      It it= a ;      It(a)
    a == b          a != b
    *a              a->m
    ++r             (void) r ++      *r ++
↑ Forward Iterator (non-const or const)
    It it ;        It ()
    r = a
    r ++
↑ Bidirectional Iterator (non-const or const)
    --r            r -              *r -
↑ Random Access Iterator (non-const or const)
    r += n         a + n           n + a
    r -= n         a - n           b - a
    a[n]
    a < b          a > b           a >= b          a <= b
```

## Algorithms

Nonmutating: `find, count, foreach, accumulate`  
 Mutating: `swap, copy, replace, remove, fill, generate, random_shuffle, sort`

## Functors (Function Objects)

Base classes: `unary_function, binary_function`  
 Arithmetic: `plus, minus, multiplies, divides`  
 Comparison: `greater, less, greater_less`  
 Logical: `logical_and, logical_or, logical_not`

## Adaptors (Functionals)

Binders (Curries) `bind1st, bind2nd`  
 Logical: `not1, not2`  
 Pointer adaptors: `ptr_fun, mem_fun_ref_t`

## Allocators

### Questions for Discussion

1. When is an algorithm performed by a container method, and when by a non-member function?
2. Why are there separate specifications for **X u(a)** and **X u=a** ? (This question is deeper than it looks!)
3. How and why do iterators use a non-OO form of inheritance?
4. How do you write the perfect binder?

<sup>1</sup> *r* is an expression of type `It&`; *a, b* are expressions of type `It`; *n* is an expression of type `iterator_traits<It>::difference_type`.