Italian C++ Community

www.italiancpp.org

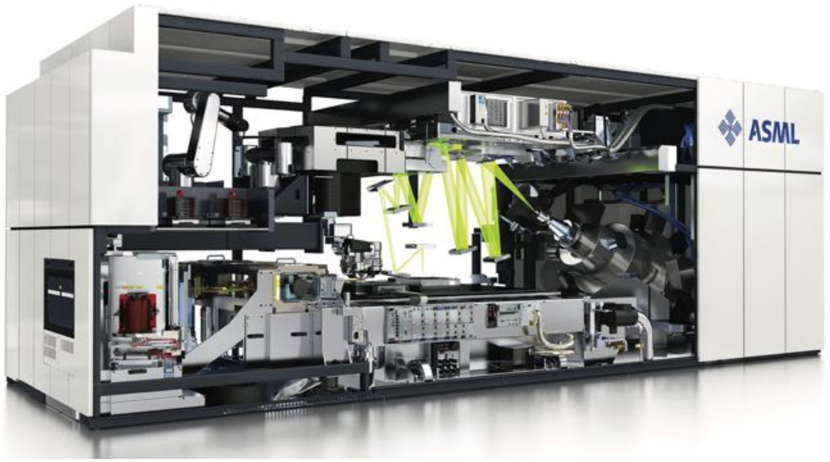
Principles of STL algorithms design

Francesco Fucci

C++ Day 2019
November 30, Parma

About Me

- Past life:
  - PhD in Computer and Automation Engineering @ Federico II Napoli
- Then:
  - Railways
  - Fusion **ITER** project as Software Engineer (www.iter.org).
  - Recently I joined **ASML** the largest producer of lithography machines in the world as Software Designer (www.asml.com)



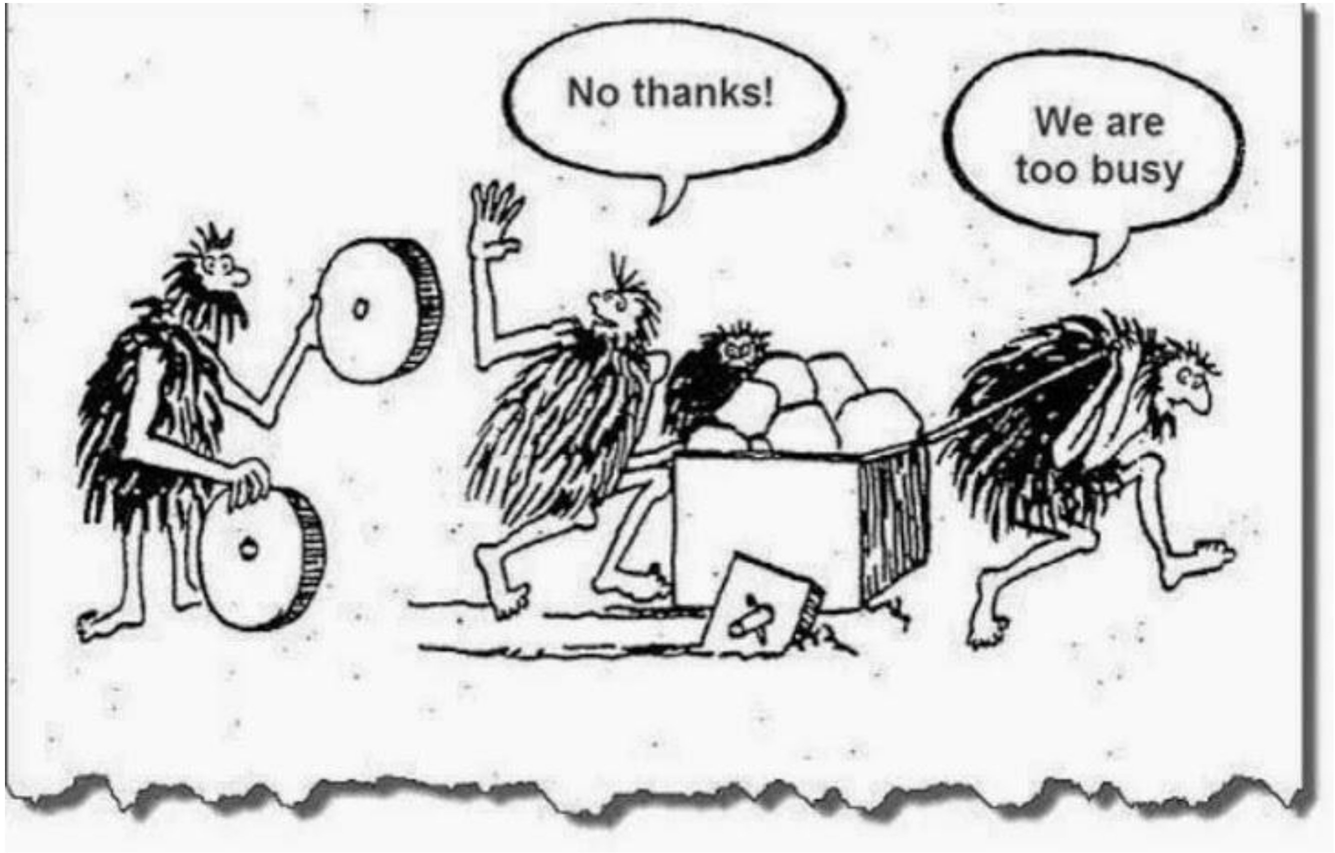
Why?


I'LL ASK WALLY TO WRITE THIS SOFTWARE.
Dilbert.com
@ScottAdamsSays
WALLY
7-30-16 © 2016 Scott Adams, Inc. /Dist. by Universal Uclick
I HAVEN'T SEEN HIM SINCE HE TOOK THAT AGILE PROGRAMMING CLASS.
WALLY

Why?

-> **Code quality?**

-> **Reinventing the wheel?**



Why?

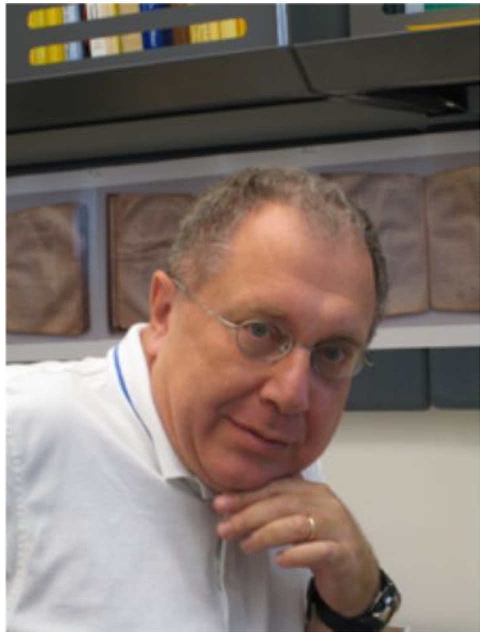
KNOW YOUR ALGORITHMS!

Why?

KNOW YOUR ALGORITHMS!

…NO RAW LOOPS MANTRA

Why?



In principle every while loop is a missed opportunity for abstraction

Disclaimer

An in-depth analysis of this topic requires more time than I have in this talk, though I try to cover the most fundamental parts

Goals

- Systematic organization of algorithms and data structure
- “Universal” representations of algorithms
- Using whole-part value semantics for data structures
- Using *abstractions of addresses* as the interface between algorithms and data structures

Goals

- Systematic organization of algorithms and data structure
- “Universal” representations of algorithms
- Using whole-part value semantics for data structures
- Using *abstractions of addresses* as the interface between algorithms and data structures

Algorithms

Functions in <algorithm>

Non-modifying sequence operations:

all_of C++11	Test condition on all elements in range (function template)
any_of C++11	Test if any element in range fulfills condition (function template)
none_of C++11	Test if no elements fulfill condition (function template)
for_each	Apply function to range (function template)
find	Find value in range (function template)
find_if	Find element in range (function template)
find_if_not C++11	Find element in range (negative condition) (function template)
find_end	Find last subsequence in range (function template)
find_first_of	Find element from set in range (function template)
adjacent_find	Find equal adjacent elements in range (function template)
count	Count appearances of value in range (function template)
count_if	Return number of elements in range satisfying condition (function template)
mismatch	Return first position where two ranges differ (function template)
equal	Test whether the elements in two ranges are equal (function template)
is_permutation C++11	Test whether range is permutation of another (function template)
search	Search range for subsequence (function template)
search_n	Search range for elements (function template)

Algorithms

Modifying sequence operations:

copy	Copy range of elements (function template)
copy_n C++11	Copy elements (function template)
copy_if C++11	Copy certain elements of range (function template)
copy_backward	Copy range of elements backward (function template)
move C++11	Move range of elements (function template)
move_backward C++11	Move range of elements backward (function template)
swap	Exchange values of two objects (function template)
swap_ranges	Exchange values of two ranges (function template)
iter_swap	Exchange values of objects pointed to by two iterators (function template)
transform	Transform range (function template)
replace	Replace value in range (function template)
replace_if	Replace values in range (function template)
replace_copy	Copy range replacing value (function template)
replace_copy_if	Copy range replacing value (function template)
fill	Fill range with value (function template)
fill_n	Fill sequence with value (function template)
generate	Generate values for range with function (function template)
generate_n	Generate values for sequence with function (function template)

Algorithms

Sorting:

sort	Sort elements in range (function template)
stable_sort	Sort elements preserving order of equivalents (function template)
partial_sort	Partially sort elements in range (function template)
partial_sort_copy	Copy and partially sort range (function template)
is_sorted C++11	Check whether range is sorted (function template)
is_sorted_until C++11	Find first unsorted element in range (function template)
nth_element	Sort element in range (function template)

Binary search (operating on partitioned/sorted ranges):

lower_bound	Return iterator to lower bound (function template)
upper_bound	Return iterator to upper bound (function template)
equal_range	Get subrange of equal elements (function template)
binary_search	Test if value exists in sorted sequence (function template)

Merge (operating on sorted ranges):

merge	Merge sorted ranges (function template)
inplace_merge	Merge consecutive sorted ranges (function template)
includes	Test whether sorted range includes another sorted range (function template)
set_union	Union of two sorted ranges (function template)
set_intersection	Intersection of two sorted ranges (function template)
set_difference	Difference of two sorted ranges (function template)
set_symmetric_difference	Symmetric difference of two sorted ranges (function template)

Algorithms Overview

Map/Transform

transform	applies a function to a range of elements, storing results in a destination range (function template)

Reduce/Accumulate (foldable)

accumulate	sums up a range of elements (function template)

Permutations

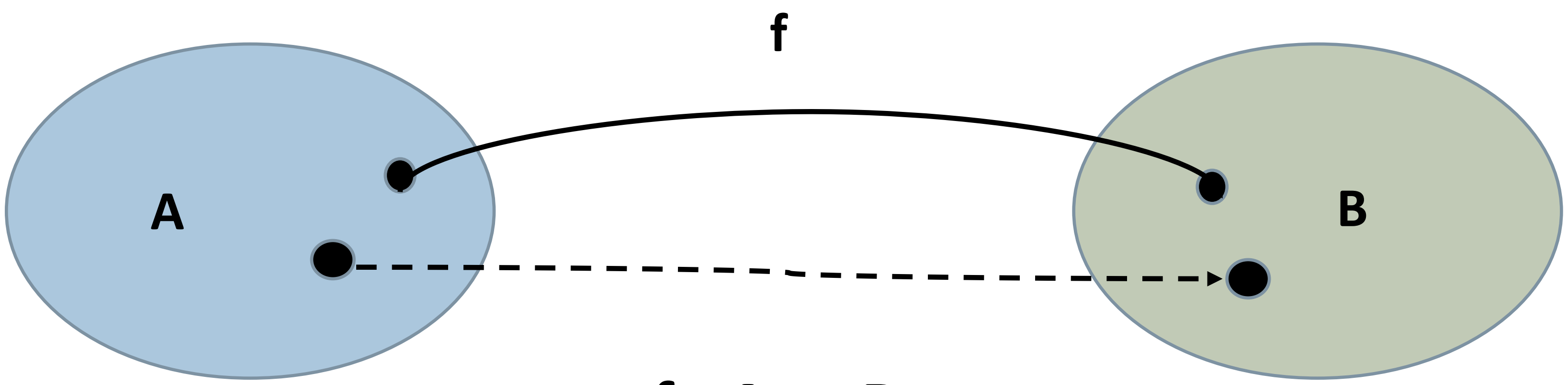
reverse	reverses the order of elements in a range (function template)
reverse_copy	creates a copy of a range that is reversed (function template)
rotate	rotates the order of elements in a range (function template)
rotate_copy	copies and rotate a range of elements (function template)

Partitioning operations

Defined in header <algorithm>

is_partitioned (C++11)	determines if the range is partitioned by the given predicate (function template)
partition	divides a range of elements into two groups (function template)

map operator



f : A -> B

<u>f transforms an element **a** of **A** into an element of **b** of **B**</u>

In Haskell for embellished types we have also **fmap**, that is out of scope for this talk

transform (C++/Haskell)

C++

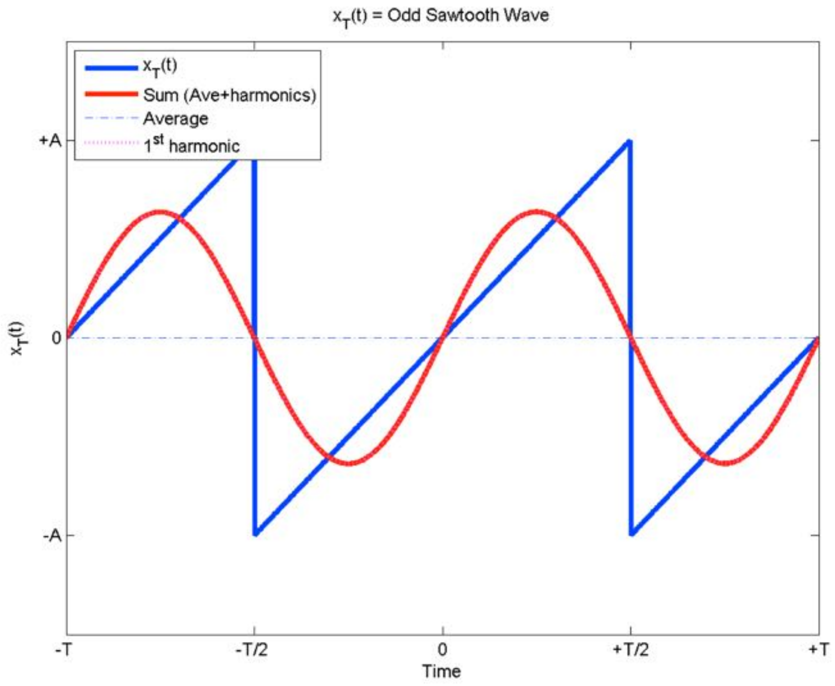
```
template<class InputIt, class OutputIt, class UnaryOperation>
OutputIt transform(InputIt first1, InputIt last1, OutputIt d_first,
                   UnaryOperation unary_op)
{
    while (first1 != last1) {
        *d_first++ = unary_op(*first1++);
    }
    return d_first;
}
```

Haskell

```
map' :: (a -> b)-> [a] -> [b]
map' f [] = []
map' f (x:xs) = f x : map f xs
```

example

Compute k coefficient of the Fourier series for a sawtooth



$$b_n = -\frac{2A}{\pi n}(-1)^n$$

A = 1

https://godbolt.org/z/VmtZPf

example (C++)

$$b_n = -\frac{2A}{\pi n}(-1)^n$$

```
std::vector<float> compute_coeff(int k){
    std::vector<float> series;
    series.resize(k);
    auto coeff_series = [](const int k) { return pow(-1,k+1) *
    (2/(M_PI*k)); };
    //Generate the series {1,k}
    std::iota(std::begin(series),std::end(series),1);
    //Map using the lambda expression {1,k}
    std::transform(std::begin(series),std::end(series),std::begin(series)
    ,coeff_series);
    return series;
}
```

example (Haskell)

$$b_n = -\frac{2A}{\pi n}(-1)^n$$

```
compute_coeff :: Int -> [Float]
compute_coeff k = map coeff_expr [1..k]
                  where coeff_expr x
= -(2/pi)*(-1)^x/(fromIntegral x)
```

Spoiler

The next is one slide of theory

concepts/typeclasses

"A concept is a way of describing a family of related object types."

Alexander A. Stepanov. "From Mathematics to Generic Programming."

```
template<typename T> concept
Hashable = requires(T a) {

{
 std::hash<T>{}(a) } ->
std::convertible_to<std::size_t>;
};
```

A typeclass is a sort of interface that defines some behavior. If a type is a part of a typeclass, that means that it supports and implements the behavior the typeclass describes. (LYAH)

```
class Eq a where
    (==), (/=) :: a -> a -> Bool

    x /= y = not (x == y)
```

example (C++)

$$b_n = -\frac{2A}{\pi n}(-1)^n$$

```
std::vector<float> compute_coeff(int k){
    std::vector<float> series;
    series.resize(k);
    auto coeff_series = [](const int k) { return pow(-1,k+1) *
    //Generate the series {1,k}
    std::iota(std::begin(series),std::end(series),1);
    //Map using the lambda expression {1,k}
    std::transform(std::begin(series),std::end(series),std::begin(series)
    ,coeff_series);
    return series;
```

example (C++)

$$b_n = -\frac{2A}{\pi n}(-1)^n$$

```
#define Container typename
#define Function typename

template<Container C, Function F>
auto process_sequence(C c,F f){
    C result(c);
    std::iota(std::begin(result),std::end(result),1);
    //Map using the lambda expression {1,k}
    std::transform(std::begin(result),std::end(result),std::begin(r
    esult),f);
    return result;
}
```

https://godbolt.org/z/6GGQKs

example (C++)

$$b_n = -\frac{2A}{\pi n}(-1)^n$$

```
std::vector<float> compute_coeff(int k){
   std::vector<float> series;
   series.resize(k);
   auto coeff_series = [](const int k) { return pow(-
   1,k+1) * (2/(M_PI*k)); };
   //Generate the series {1,k}
   return process_sequence(series,coeff_series);
}
```

example (C++)

$$b_n = -\frac{2A}{\pi n}(-1)^n$$

```
std::forward_list<float> compute_coeff(int k){
   std::forward_list<float> series(k);
   auto coeff_series = [](const int k) { return pow(-
   1,k+1) * (2/(M_PI*k)); };
   //Generate the series {1,k}
   return process_sequence(series,coeff_series);
}
```

example (C++)

$$b_n = -\frac{2A}{\pi n}(-1)^n$$

```
template<ForwardIt It, SequenceExpression S>
void process_sequence_it(It first, It last,S expression) {
    if(first == last){
        return ;
    }
    //Generate the series {1,k}
    std::iota(first, last,1);
    std::transform(first,last,first,expression);
}
```

https://godbolt.org/z/Zh5HJD

example (C++)

$$b_n = -\frac{2A}{\pi n}(-1)^n$$

```
template<Container C>
C compute_coeff(int k){
   C series(k);
   auto coeff_series = [](const int k) { return pow(-
   1,k+1) * (2/(M_PI*k)); };
   process_sequence_it(std::begin(series),std::end(se
   ries),coeff_series);
   return series;
}
```

https://godbolt.org/z/Zh5HJD

example (C++)

$$b_n = -\frac{2A}{\pi n}(-1)^n$$

This should be a concept

```
template<Container C>
C compute_coeff(int k){
   C series(k);
   auto coeff_series = [](const int k) { return pow(-
   1,k+1) * (2/(M_PI*k)); };
   process_sequence_it(std::begin(series),std::end(se
   ries),coeff_series);
   return series;
}
```

https://godbolt.org/z/Zh5HJD

example (C++)

$$b_n = -\frac{2A}{\pi n}(-1)^n$$

```
auto v = compute_coeff<std::vector<float>>(10);
auto l = compute_coeff<std::forward_list<float>>(10);
std::copy(std::begin(v),std::end(v),std::ostream_iter
ator<float>{std::cout,","});
std::copy(std::begin(l),std::end(l),std::ostream_iter
ator<float>{std::cout,","});
```

https://godbolt.org/z/Zh5HJD

partition

Defined in header <algorithm>

`template< class BidirIt, class UnaryPredicate >` `BidirIt partition( BidirIt first, BidirIt last, UnaryPredicate p );`		(until C++11)
`template< class ForwardIt, class UnaryPredicate >` `ForwardIt partition( ForwardIt first, ForwardIt last, UnaryPredicate p );`	(1)	(since C++11) (until C++20)
`template< class ForwardIt, class UnaryPredicate >` `constexpr ForwardIt partition( ForwardIt first, ForwardIt last, UnaryPredicate p );`		(since C++20)

`template< class ExecutionPolicy, class ForwardIt, class UnaryPredicate >`
`ForwardIt partition( ExecutionPolicy&& policy,`
`ForwardIt first, ForwardIt last, UnaryPredicate p );`
	(2)	(since C++17)

1) Reorders the elements in the range `[first, last)` in such a way that all elements for which the predicate p returns `true` precede the elements for which predicate p returns `false`. Relative order of the elements is not preserved.

2) Same as (1), but executed according to `policy`. This overload does not participate in overload resolution unless `std::is_execution_policy_v<std::decay_t<ExecutionPolicy>>` is true

Parameters

`first, last` - the range of elements to reorder

`policy` - the execution policy to use. See execution policy for details.

`p` - unary predicate which returns `true` if the element should be ordered before other elements.

The expression `p(v)` must be convertible to `bool` for every argument v of type (possibly const) VT, where VT is the value type of `ForwardIt`, regardless of value category, and must not modify v. Thus, a parameter type of `VT&` is not allowed, nor is `VT` unless for VT a move is equivalent to a copy (since C++11).

partition

DEFINED ON FORWARD ITERATORS

```
template<class ForwardIt, class UnaryPredicate>
ForwardIt partition(ForwardIt first, ForwardIt last,
UnaryPredicate p) {
        auto first = std::find_if_not(first, last, p);
        if (first == last) return first;
        for (ForwardIt i = std::next(first); i != last; ++i) {
                if (p(*i)) {
                        std::iter_swap(i, first);
                        ++first;
                }
        }
        return first;
}
```

ONLY DEFINED ON LIST

```
stable_partition' :: (a -> Bool) -> [a] ->
[a]
stable_partition' p [] = []
stable_partition' p x = (filter p x) ++
(filter (not . p) x)
```

partition: example

Separate items within an interval with items outside an interval

```
template <typename C>
auto partition_in_a_range(C&& c, const
std::pair<float,float>& range){
    auto filter_cond = [r = range](float x){ return x
    >= r.first && x <= r.second; };
    auto it =
    std::partition(std::begin(c),std::end(c),filter_co
    nd);
    return it;
}
```

partition (C++/Haskell)

A vector of numbers separate the elements belonging to a range in two subsets

```
std::vector<float>
v{0.5f,0.2f,0.1f,2.4f,2.2f,1.2f,1.3f,1.4f,2.0f,4.5f};
std::pair<float,float> r = {1.1f,1.5f};
auto partition_point = partition_in_a_range(v,r);
```

```
stable_partition' (\x -> (x >= 1.1 && x <= 1.5))
[0.5,0.2,0.1,2.4,2.2,1.2,1.3,1.4,2.0,4.5]
```

partition

A linked list of numbers separate the elements belonging to a range, in two subsets

```
std::forward_list<float> v;
v =
{0.5f,0.2f,0.1f,2.4f,2.2f,1.2f,1.3f,1.4f,2.0f,4.5f};
std::pair<float,float> r = {1.1f,1.5f};
auto partition_point = partition_in_a_range(v,r);
std::copy(std::begin(v),partition_point,std::ostream_i
terator<float>{std::cout,","});
```

reduce

```
template <class InputIterator, class BinaryOperation> typename
iterator_traits<InputIterator>::value_type
reduce(InputIterator first, InputIterator last, BinaryOperation op) {
   if (first == last) {
       return identity_element(op);
   }
   typename iterator_traits<InputIterator>::value_type
   result = *first;
   while (++first != last)  result = op(result, *first);
   return result;
}
```

https://godbolt.org/z/UHzkGX

reduce

```
template <class InputIterator, class BinaryOperation>
typename iterator_traits<InputIterator>::value_type
reduce(InputIterator first, InputIterator last,
BinaryOperation op) {
    if (first == last) {
        return identity_element(op);
    }
    typename iterator_traits<InputIterator>::value_type
    result = *first;
    while (++first != last)  result = op(result, *first);
    return result;
}
```

OPERATOR IS ASSOCIATIVE

reduce

```
template <class InputIterator, class BinaryOperation>
typename iterator_traits<InputIterator>::value_type
reduce(InputIterator first, InputIterator last,
BinaryOperation op) {
    if (first == last) {
        return identity_element(op);
    }
    typename iterator_traits<InputIterator>::value_type
    result = *first;
    while (++first != last)  result = op(result, *first);
    return result;
}
```

reduce

```
std::forward_list<float> l = {4.0,7.0,3.2,1.1,23.5,0.8};
auto _max = [](const auto a, const auto b){ return std::max(a,b); };
auto _min = [](const auto a, const auto b){ return std::min(a,b); };
auto result = reduce(std::begin(l),std::end(l),_max);
std::cout << "max: " << result << '\n';
result = reduce(std::begin(l),std::end(l),_min);
std::cout << "min: " << result << '\n';
```

reduce (Haskell): foldable

In Haskell we have two foldable operations:

- **Right folding:** folding happens on the right side
- **Left folding:** folding happens on the left side

```
foldr :: (a -> b -> b) -> b -> [a] -> b

foldr f v []       = v
foldr f v (x:xs) = f x (foldr f v xs)
```

reduce (Haskell): foldable

In Haskell we have two foldable operations:

- **Right folding:** folding happens on the right side
- **Left folding:** folding happens on the left side

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f v []       = v
foldr f v (x:xs) = f x (foldr f v xs)
```

```
l = [4.0,7.0,3.2,1.1,23.5,0.8]
foldr max 0.0 l
foldr min 30.0 l
```

Thank you!

References

Alex Stepanov: Algorithmic journeys

https://www.youtube.com/user/A9Videos/playlists

Sean Parent: C++ seasoning 2013 (No Raw Loops)

https://www.youtube.com/watch?v=W2tWOdzgXHA&t=3284s

Ivan Cuckic

https://www.youtube.com/watch?v=FxcT4vK01-w

Haskell

http://learnyouahaskell.com/

Backup

lower_bound, upper_bound

Binary search is based on the Intermediate Value Theorem (Bolzano-Cauchy)

```
template <ForwardIterator I, Predicate P>
I partition_point_n(I f, DifferenceType<I>
n, P p) {
    while (n) {
        I middle(f);
        DifferenceType<I> half(n >> 1);
        advance(middle, half);
        if (!p(*middle)) {
            n = half;
        } else {
            f = ++middle;
            n = n - (half + 1);
        }
    }
```

```
template <ForwardIterator I>
I lower_bound(I f, I l, ValueType<I> a) {
    return partition_point(f, l,
                               [=](ValueType<I>
x) { return x < a; });
}
```

```
template <ForwardIterator I>
I upper_bound(I f, I l, ValueType<I> a) {
    return partition_point(f, l,
                               [=](ValueType<I>
x) {return x <= a;});
}
```

Alexander A. Stepanov. “From Mathematics to Generic Programming

Categories / STL

- Algorithms are affiliated with mathematical theories

- **STL defines “orthogonal” structure based on functionality**

- **Haskell:** Algorithms are specialized for a type (also for a parametric type)

Categorical Theories versus STL

For a long time, people believed that only categorical theories were good for programming. When the C++ Standard Template Library (STL) was first proposed, some computer scientists opposed it on the grounds that many of its fundamental concepts, such as **Iterator**, were underspecified. In fact, it is this underspecification that gives the library its generality. While linked lists and arrays are not computationally isomorphic, many STL algorithms are defined on input iterators and work on both data structures. If you can get by with fewer axioms, you allow for a wider range of implementations.

Alexander A. Stepanov. “From Mathematics to Generic Programming

Iterator types

Haskell has no iterator concept

Iterator types

"An iterator is a concept used to express where we are in a sequence. To be an iterator, a type must support three operations:"

Iterator types

Regular type operations: default constructor, copy constructor, move constructor, destructor, total ordering

Successor: Next object in the sequence

Dereference: Access the object of the iterator

Haskell has no iterator concept

Iterator types

- **Input iterators:** defined on streams, one directional traversal. Once the object is consumed is gone
- **Forward iterators :** iterators can move just in one direction, the traversal can be repeated multiple times
- **Bidirectional Iterators:** iterators can move in two directions
- **Random-access iterators:** the access time is independent from the location of the object