

A quick overview of the Standard Template Library

Advanced Programming and Algorithmic Design

Alberto Sartori

November 30, 2017

Outline

- 1 Iterators
- 2 Containers

1 Iterators

2 Containers

What is an Iterator?

Design pattern

Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

A generalization of a pointer

- indirect access (`operator*()`, `operator->()`)
- operations for moving to point to a new element (`operator++()`, `operator--()`)

Iterators in the STL

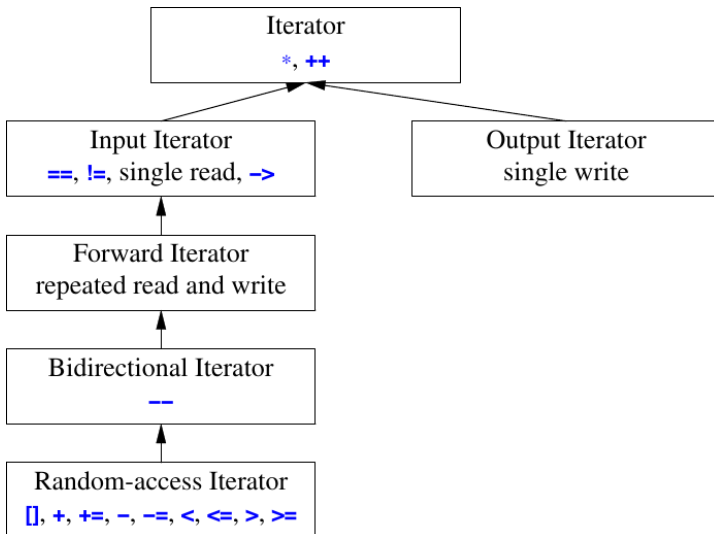
Their role

- Iterators are the glue that ties the standard-library algorithms to their data
- Iterators are the mechanism used to minimize an algorithm's dependence on the data structures on which it operates.

Alex Stepanov

The reason that STL containers and algorithms work so well together is that they know nothing of each other.

Iterator categories



Does our iterator work?

```
template <typename T>
class List<T>::Iterator {
    ...
};
```

Does our iterator work?

```
#include <iterator>
```

```
...
```

```
template <typename T>
```

```
class List<T>::Iterator : public
```

```
    std::iterator<std::forward_iterator_tag, T> {
```

```
    ...
```

```
};
```



```
template <typename Cat ,  
          typename T,  
          typename Dist = ptrdiff_t ,  
          typename Ptr = T*,  
          typename Ref = T&>  
struct iterator{  
    using value_type = T;  
    using difference_type = Dist;  
    using pointer = Ptr;  
    using reference = Ref;  
    using iterator_category = Cat;  
};
```

- 1 Iterators
- 2 Containers

Containers

Definition

A container holds a sequence of objects.

Two categories

- Sequence containers: provide access to sequences of elements
- Associative containers: provide associative lookup based on a key

Associative containers

- Ordered
- Unordered

Sequence containers

Sequence Containers

vector<T,A>	A contiguously allocated sequence of T s; the default choice of container
list<T,A>	A doubly-linked list of T ; use when you need to insert and delete elements without moving existing elements
forward_list<T,A>	A singly-linked list of T ; ideal for empty and very short sequences
deque<T,A>	A double-ended queue of T ; a cross between a vector and a list; slower than one or the other for most uses

Ordered associative containers

Ordered Associative Containers (§iso.23.4.2)

C is the type of the comparison; **A** is the allocator type

<code>map<K,V,C,A></code>	An ordered map from K to V ; a sequence of (K , V) pairs
<code>multimap<K,V,C,A></code>	An ordered map from K to V ; duplicate keys allowed
<code>set<K,C,A></code>	An ordered set of K
<code>multiset<K,C,A></code>	An ordered set of K ; duplicate keys allowed

Unordered associative containers

Unordered Associative Containers (§iso.23.5.2)

H is the hash function type; **E** is the equality test; **A** is the allocator type

`unordered_map<K,V,H,E,A>`

An unordered map from **K** to **V**

`unordered_multimap<K,V,H,E,A>`

An unordered map from **K** to **V**; duplicate keys allowed

`unordered_set<K,H,E,A>`

An unordered set of **K**

`unordered_multiset<K,H,E,A>`

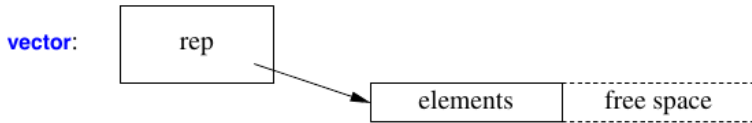
An unordered set of **K**; duplicate keys allowed

Array

array:

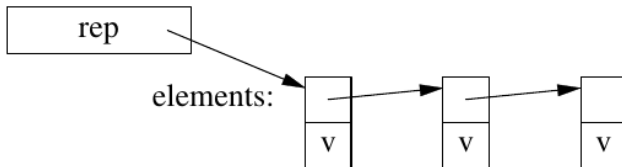
elements

Vector

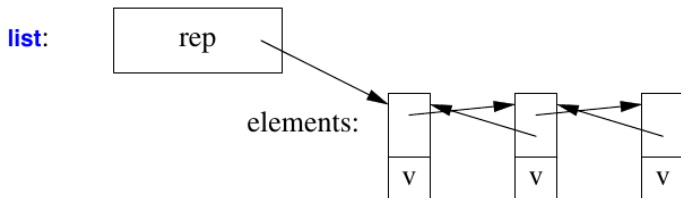


Forward list

forward_list:

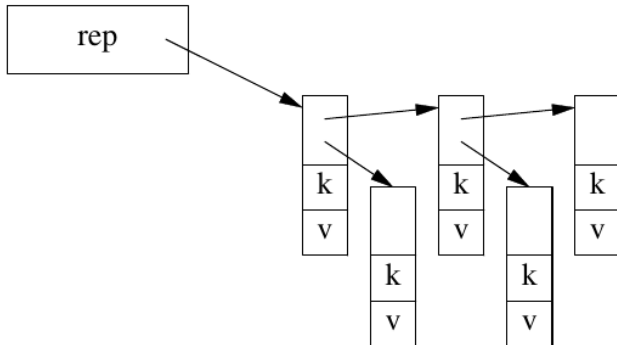


List

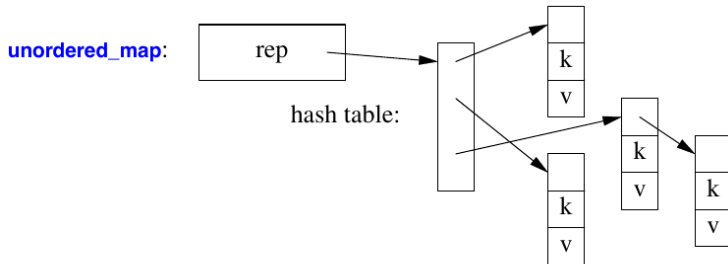


Map

map:



Unordered map



Operations and types

Container:

value_type, size_type, difference_type, pointer, const_pointer, reference, const_reference
 iterator, const_iterator, ?reverse_iterator, ?const_reverse_iterator, allocator_type
 begin(), end(), cbegin(), cend(), ?rbegin(), ?rend(), ?crbegin(), ?crend(), =, ==, !=
 swap(), ?size(), max_size(), empty(), clear(), get_allocator(), constructors, destructor
 ?<, ?<=, ?>, ?>=, ?insert(), ?emplace(), ?erase()

Sequence container:

assign(), front(), resize()
 ?back(), ?push_back()
 ?pop_back(), ?emplace_back()

Associative container:

key_type, mapped_type, ?[], ?at()
 lower_bound(), upper_bound(), equal_range()
 find(), count(), emplace_hint()

push_front(), pop_front()
 emplace_front()

[], at()
 shrink_to_fit()

Ordered container:

key_compare
 key_comp()
 value_comp()

Hashed container:

key_equal(), hasher
 hash_function()
 key_equal()
 bucket interface

List:

remove()
 remove_if(), unique()
 merge(), sort()
 reverse()

deque

data()
 capacity()
 reserve()

vector

splice()

insert_after(), erase_after()
 emplace_after(), splice_after()

list

forward_list

map

multimap

set

unordered_map

multiset

unordered_set

unordered_multimap

unordered_multiset

Operation complexity

Standard Container Operation Complexity					
	[] §31.2.2	List §31.3.7	Front §31.4.2	Back §31.3.6	Iterators §33.1.2
vector	const	$O(n)+$		const+	Ran
list		const	const	const	Bi
forward_list		const	const		For
deque	const	$O(n)$	const	const	Ran
stack				const	
queue			const	const	
priority_queue			$O(\log(n))$	$O(\log(n))$	
map	$O(\log(n))$	$O(\log(n))+$			Bi
multimap		$O(\log(n))+$			Bi
set		$O(\log(n))+$			Bi
multiset		$O(\log(n))+$			Bi
unordered_map	const+	const+			For
unordered_multimap		const+			For
unordered_set		const+			For
unordered_multiset		const+			For
string	const	$O(n)+$	$O(n)+$	const+	Ran
array	const				Ran
built-in array	const				Ran
valarray	const				Ran
bitset	const				

Prime numbers

```
#include <vector>

int main(){
    std::vector<int> primes;

    primes.emplace_back(2);

    for (int i=3; i<=max; ++i)
        if (is_prime(i))
            primes.emplace_back(i);

    for (const auto& x: primes)
        std::cout << x << std::endl;
}
```

Word count

```
#include <map>

int main(){
    std::map<std::string, int> words;

    for (std::string s; std::cin>>s;)
        ++words[s];

    for (const auto& x: words)
        std::cout << x.first << ": "
                    << x.second << std::endl;
}
```


Word count

```
#include <map>

int main(){
    std::unordered_map<std::string, int> words;

    for (std::string s; std::cin>>s;)
        ++words[s];

    for (const auto& x: words)
        std::cout << x.first << ": "
                    << x.second << std::endl;
}
```

Thank you for your kind attention!