

C++ Standard Library Algorithms

Piotr Nycz (Nokia)

09-03-2020

Basic information

<https://en.cppreference.com/w/cpp/algorithm>

`<algorithm>`

Non-modifying sequence

(all_of, find_if, ...)

Modifying sequence

(copy, remove, ...)

Partitioning

Sorting

Permutation

Binary Search

lower/upper_bound, ...

Set/Heap operations

Min/max

Comparison

equal

lexographical_compare, ...

`<numeric>`

iota

accumulate

reduce

transform_reduce

inner_product

adjacent_difference

partial_sum

inclusive_scan

exclusive_scan

transform_inclusive_scan

transform_exclusive_scan

`<memory>`

uninitialized_copy, ...

destroy, ...

`<iterator>`

Adaptors

make_move_iterator,

make_reverse_iterator, ...

back_inserter, ...

Stream iterators

istream_iterator,

ostream_iterator, ...

Operations

begin, end, rbegin, ...

size, empty, data, ...

distance, next, prev, ...

Basic information – in-class algorithms

<https://en.cppreference.com/w/cpp/container>

Associative containers

$O(\log N)$

count
find
equal_range
lower_bound
upper_bound

Unordered associative containers

$O(1)$

count
find

Most containers

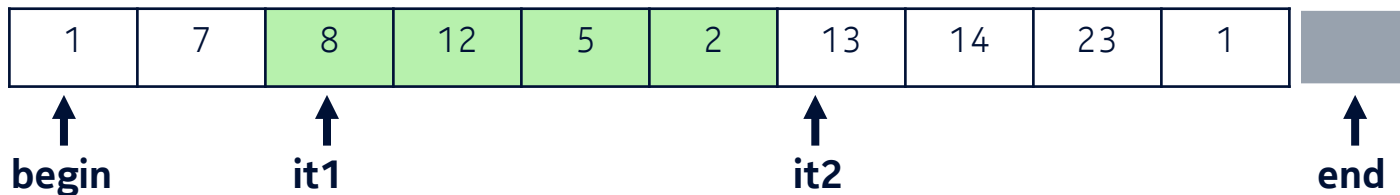
erase
swap

erase_if (C++20)

1. Provide better performance than standalone algorithms (like `std::find` ($O(N)$))
2. Can modify containers
 1. standalone algorithms can modify only elements
 2. compare behavior of `std::remove` and `std::vector<T>::erase`

Ranges in C++17

- Range defined by 2 iterators [it1, it2) is the basic input to all algorithms
 - Range contains all element between it1(inclusively) and it2 (exclusively!)

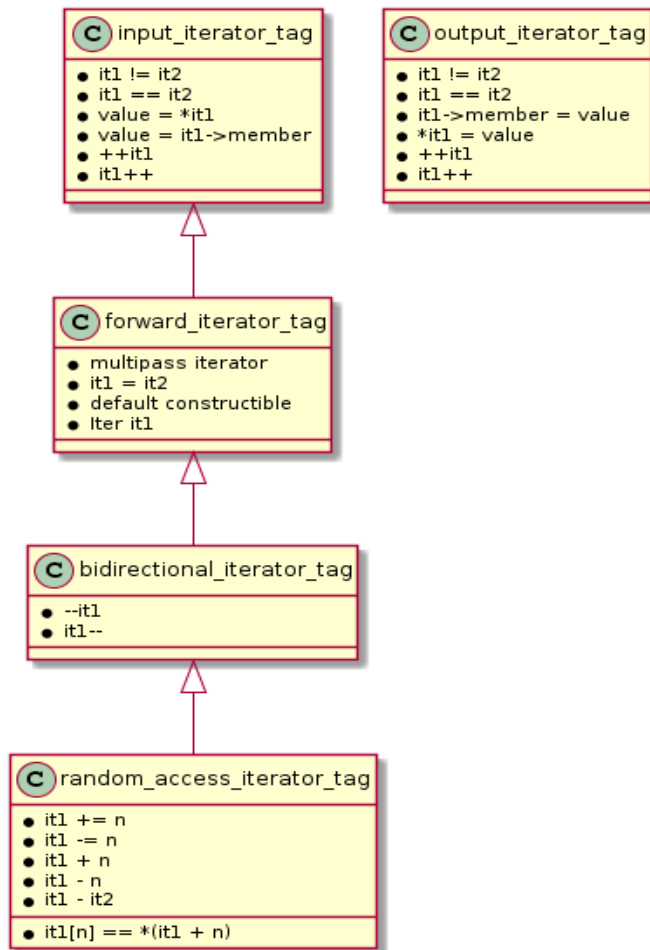


- Iterator can be anything that points to some element
 - “C” Pointer (**T***) is a valid iterator (when points to elements of an array **T[N]**)
 - All Standard Library containers have corresponding iterators (access via begin/end methods)
 - Full range for container (like **std::vector<int> a;**) is (**a.begin()**, **a.end()**)
 - Full range for “C” array (like **char b[] = “ala ma kota”;**) is (**b**, **b + N**)
 - Since C++11 both containers and arrays ranges can be defined by functions begin, end: (**std::begin(x)**, **std::end(x)**)
 - Iterator can be also something not related to container nor array – like stream iterators
- An example – printing all elements of a **vector<int> v;** – by copying to **std::cout**:

```
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, ""));
```

About performance

- Algorithms are written with performance in mind
 - But compiler optimization must be ON – due to big number of inline functions
- Algorithms have various versions aligned to type of iterators (see on right →). This is to achieve the best performance for the given iterator/container type. Simple example of that is **std::distance**, **std::advance**



About performance - multithreading

((from gcc9.1))

Some algorithms can be run in multithreads modes

- **namespace std::execution { sequenced_policy seq; }**
 - Default - operations are performed in sequence (like in default pre-C++17 mode)
- **namespace std::execution { parallel_policy par; }**
 - Operations might be performed in parallel (in different threads)
 - It requires from user to ensure no data races happen
- **namespace std::execution { parallel_unsequenced_policy par_unseq; }**
 - Operations might be performed in parallel, vectorized, migrated from thread to thread
 - It requires vectorization-safe code – e.g. no mutex allowed
 - But still no data races allowed – so it is really hard to use. But it is the most promising.

NOKIA

Copyright and confidentiality

The contents of this document are proprietary and confidential property of Nokia. This document is provided subject to confidentiality obligations of the applicable agreement(s).

This document is intended for use of Nokia's customers and collaborators only for the purpose for which this document is submitted by Nokia. No part of this document may be reproduced or made available to the public or to any third party in any form or means without the prior written permission of Nokia. This document is to be used by properly trained professional personnel. Any use of the contents in this document is limited strictly to the use(s) specifically created in the applicable agreement(s) under which the document is submitted. The user of this document may voluntarily provide suggestions, comments or other feedback to Nokia in respect of the contents of this document ("Feedback").

Such Feedback may be used in Nokia products and related specifications or other documentation. Accordingly, if the user of this document gives Nokia Feedback on the contents of this document, Nokia may freely use, disclose, reproduce, license, distribute and otherwise commercialize the feedback in any Nokia product, technology, service, specification or other documentation.

Nokia operates a policy of ongoing development. Nokia reserves the right to make changes and improvements to any of the products and/or services described in this document or withdraw this document at any time without prior notice.

The contents of this document are provided "as is". Except as required by applicable law, no warranties of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose,

are made in relation to the accuracy, reliability or contents of this document. NOKIA SHALL NOT BE RESPONSIBLE IN ANY EVENT FOR ERRORS IN THIS DOCUMENT or for any loss of data or income or any special, incidental, consequential, indirect or direct damages howsoever caused, that might arise from the use of this document or any contents of this document.

This document and the product(s) it describes are protected by copyright according to the applicable laws.

Nokia is a registered trademark of Nokia Corporation. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.