

Document number: P3331R1

Project: Programming Language C++

Audience: Library Evolution Working Group, Library Working Group

Nikita Sakharin, PJSC Sberbank <[nikitasa1997@gmail.com](mailto:nikitasa1997@gmail.com)>

Date: 2025-02-04

# Accessing The First and Last Elements in Associative Containers

## Contents

0. [Revision history](#)
  - [Changes since P3331R0](#)
1. [Abstract](#)
2. [Motivation](#)
  1. [Intuitive name for cumbersome expression](#)
  2. [Member function contains added in C++20](#)
  3. [Java and Rust](#)
3. [Design considerations](#)
  1. [Member functions front\(\) and back\(\) for unordered associative containers](#)
  2. [Naming scheme](#)
    1. [first/last](#)
    2. [min/max](#)
    3. [front/back](#)
4. [Questions for Committee](#)
5. [Wording](#)
  1. [Associative container requirements](#)
  2. [Associative containers](#)
  3. [Container adaptors](#)
6. [Implementation Experience](#)
7. [Acknowledgements](#)
8. [Reference](#)
  - [StackOverflow](#)
  - [Proposal](#)
  - [Java](#)
  - [Rust](#)

## 0. Revision history

### Changes since [P3331R0](#)

- Improve motivation;
- Fix typo.

## 1. Abstract

This paper proposes to add two member functions to associative containers (and adaptors that conform to such requirements):

- `front()`: get the first element in container;
- `back()`: get the last element in container.

The effect of calling `front()` or `back()` for an empty container is undefined.

## 2. Motivation

### 2.1. Intuitive name for cumbersome expression

There are several popular questions about C++ on **StackOverflow**:

- “Getting first value from map in C++”<sup>[1]</sup>
- “Last key in a `std::map`”<sup>[2]</sup>
- “Why no `front()` method on `std::map` (and other associative containers from the STL)?”<sup>[3]</sup>
- “Is there a design reason why `std::set` doesn't have `front` and `back` member functions?”<sup>[4]</sup>

To illustrate the problem, suppose we have a `std::map` (or `std::multimap`) declared as follows:

```
std::map<Key, T> m;
```

The table below shows access to the first and last elements, keys, and values of the container `m`. It compares the existing solutions with those that use the proposed member functions:

Expression		Semantics	
existing	proposed		
<code>*m.begin()</code>	<code>m.front()</code>	first	element
<code>m.begin()-&gt;first</code>	<code>m.front().first</code>		key
<code>m.begin()-&gt;second</code>	<code>m.front().second</code>		value
<code>*m.rbegin()</code> <code>*prev(m.end())</code> <code>*--m.end()</code>	<code>m.back()</code>	last	element
<code>m.rbegin()-&gt;first</code> <code>prev(m.end())-&gt;first</code> <code>(--m.end())-&gt;first</code>	<code>m.back().first</code>		key
<code>m.rbegin()-&gt;second</code> <code>prev(m.end())-&gt;second</code> <code>(--m.end())-&gt;second</code>	<code>m.back().second</code>		value

Although the expressions in the leftmost column have already become idiomatic, they can be difficult to read and cumbersome. Also, the use of iterators seems redundant for accessing the first or last element of a container.

### 2.2. Member function `contains` added in C++20

Exactly the same reason behind adding the `contains` member function to (unordered) associative containers in C++20 was to simplify checking for an element in a container, allowing us to do so without using iterators.

There was popular question (with duplicate) on **StackOverflow** before C++20:

- “How to find if a given key exists in a std::map”<sup>[5]</sup>
- “Determine if map contains a value for a key?”<sup>[6]</sup>
- “How to check that an element is in a std::set?”<sup>[7]</sup>

Prior to C++20, the following code was often used to check for the presence of a given key in the (unordered) associative container c:

```
if (c.find(key) != c.end()) {
    // m contains an element equal to the given
}
```

There was a more elegant way, but the name of the member function was confusing:

```
if (c.count(key)) {
    // m contains an element equal to the given
}
```

The **Proposal** “Checking for Existence of an Element in Associative Containers”<sup>[8]</sup> was written by Mikhail Maltsev to address this issue. This **Proposal** was merged to C++20 and the contains member function was added to (unordered) associative containers in order to give an intuitive name and shorten the cumbersome expression:

Prior C++20	Since C++20
<pre>if (c.find(key) != c.end()) {     // m contains an element equal to the given }</pre>	<pre>if (c.contains(key)) {     // m contains an element equal to the given }</pre>
<pre>if (c.count(key)) {     // m contains an element equal to the given }</pre>	

## 2.3. Java and Rust

- The SortedSet interface in **Java** (implemented by the TreeSet class) has the first() and last() methods to get the first<sup>[9]</sup> and last<sup>[10]</sup> elements, respectively.
- Similarly to **Java**, the BTreeSet structure in **Rust** also has first() and last() methods to get the first<sup>[11]</sup> and last<sup>[12]</sup> element, respectively.

## 3. Design considerations

### 3.1 Member functions front() and back() for unordered associative containers

There is no use for front() and back() member functions for unordered associative containers (std::unordered\_map, std::unordered\_multimap, std::unordered\_set, std::unordered\_multiset). They organize their elements according to hash values rather than key order used by associative containers (std::map, std::multimap, std::set, std::multiset).

### 3.2 Naming scheme

Special attention should be paid to naming. There are 3 possible schemes described below.

### 3.2.1 first/last

In the `<algorithm>` header file, the words `first` and `last` are often used as part of function (not class member) names:

- `std::ranges::find_last`
- `std::ranges::find_last_if`
- `std::ranges::find_last_if_not`
- `std::find_first_of`
- `std::ranges::find_first_of`
- `std::ranges::fold_left_first`
- `std::ranges::fold_right_last`
- `std::ranges::fold_left_first_with_iter`

Also, in the `<string>` class, the words `first` and `last` are often used as part of member function names:

- `std::basic_string::find_first_of`
- `std::basic_string::find_first_not_of`
- `std::basic_string::find_last_of`
- `std::basic_string::find_last_not_of`

Even more often, `first` and `last` occur as function parameter names in the `<algorithm>` header, defining the beginning and end of the range to be iterated over.

As already noted, in **Java**<sup>[8,9]</sup> and **Rust**<sup>[10,11]</sup>, the corresponding methods are named exactly this way.

### 3.2.2 min/max

According to the given comparator, the first element is the smallest and the last element is the largest.

### 3.2.3 front/back

There are 6 sequence containers in **C++ STL**:

- `std::array`
- `std::deque`
- `std::forward_list`
- `std::inplace_vector`
- `std::list`
- `std::vector`

Each of these containers, with the only exception of `forward_list`, has two member functions: `front()` and `back()`.

Classes `std::basic_string`, `std::basic_string_view`, `std::span` also have these member functions.

There are 12 functions (not class members) declared in section § 25.7 [iterator.range] of the **Standard**:

- `std::begin`
- `std::end`
- `std::cbegin`
- `std::cend`
- `std::rbegin`
- `std::rend`
- `std::crbegin`
- `std::crend`

- `std::size`
- `std::ssize`
- `std::empty`
- `std::data`

These functions unify the handling of arrays in the **C** style and the containers from the **STL**. Therefore, if the **Committee** is considering expanding this section by adding the functions `std::front` and `std::back`, it would make sense to name the proposed member functions in accordance with this scheme.

## 4. Questions for Committee

1. Which naming scheme should be used?

## 5. Wording

Based on [N5001](#), assuming the third naming scheme (`front/back`) is used.

### 5.1 Associative container general requirements

Add to section § 23.2.7.1 [`associative.reqmts.general`] the following:

```
b.front()
    Result: reference; const_reference for constant b.
    Effects: Equivalent to: return *b.begin();

b.back()
    Result: reference; const_reference for constant b.
    Effects: Equivalent to: return *--b.end();
```

### 5.2 Associative containers

To each section from the list:

- § 23.4.3.1 [`map.overview`]
- § 23.4.4.1 [`multimap.overview`]
- § 23.4.6.1 [`set.overview`]
- § 23.4.7.1 [`multiset.overview`]

Add the following:

```
reference front();
const_reference front() const;

reference back();
const_reference back() const;
```

### 5.3 Container adaptors

To each section from the list:

- § 23.6.8.2 [`flat.map.defn`]
- § 23.6.9.2 [`flat.multimap.defn`]
- § 23.6.11.2 [`flat.set.defn`]
- § 23.6.12.2 [`flat.multiset.defn`]

Add the following:

```
reference front();
const_reference front() const;

reference back();
const_reference back() const;
```

## 6. Implementation Experience

The implementation of these functions is exactly the code that they are supposed to replace. To each class from the list:

- `std::map`
- `std::multimap`
- `std::set`
- `std::multiset`
- `std::flat_map`
- `std::flat_multimap`
- `std::flat_set`
- `std::flat_multiset`

Add the following:

```
reference front() {
    return *this->begin();
}

const_reference front() const {
    return *this->cbegin();
}

reference back() {
    return *--this->end();
}

const_reference back() const {
    return *--this->cend();
}
```

## 7. Acknowledgements

Many thanks to Antony Polukhin for assistance in preparation of this paper.

## 8. Reference

### StackOverflow:

1. [Getting first value from map in C++](#)
2. [Last key in a std::map](#)
3. [Why no front\(\) method on std::map \(and other associative containers from the STL\)?](#)
4. [Is there a design reason why std::set doesnt have front and back member functions?](#)
5. [How to find if a given key exists in a std::map](#)
6. [Determine if map contains a value for a key?](#)
7. [How to check that an element is in a std::set?](#)

### Proposal:

8. [Checking for Existence of an Element in Associative Containers](#)

## Java:

9. [SortedSet.first](#)
10. [SortedSet.last](#)

## Rust:

11. [BTreeSet.first](#)
12. [BTreeSet.last](#)