1. Abstract

This paper proposes to add two member functions to associative containers (and adaptors that conform to these 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 two popular questions about C++ on StackOverflow:
The following table provides answers to the questions and compares them to the code that uses the proposed member functions. We assume that the variable m used in the table has type std::map<K, T>:

<table>
<thead>
<tr>
<th>Expression</th>
<th>before</th>
<th>after</th>
<th>Semantic</th>
</tr>
</thead>
<tbody>
<tr>
<td>*m.begin()</td>
<td>m.front()</td>
<td></td>
<td>element</td>
</tr>
<tr>
<td>(*m.begin())-&gt;first</td>
<td>m.front().first</td>
<td></td>
<td>key</td>
</tr>
<tr>
<td>(*m.begin())-&gt;second</td>
<td>m.front().second</td>
<td></td>
<td>mapped</td>
</tr>
<tr>
<td>*m.rbegin()</td>
<td>m.back()</td>
<td></td>
<td>element</td>
</tr>
<tr>
<td>*prev(m.end())</td>
<td>*--m.end()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(*m.rbegin())-&gt;first</td>
<td>m.back().first</td>
<td></td>
<td>key</td>
</tr>
<tr>
<td>(*m.rbegin())-&gt;second</td>
<td>*prev(m.end())-&gt;second</td>
<td><em>(</em>--m.end())-&gt;second</td>
<td>mapped</td>
</tr>
<tr>
<td>(*m.rbegin())-&gt;second</td>
<td>m.back().second</td>
<td></td>
<td>mapped</td>
</tr>
</tbody>
</table>

Although the expressions in the leftmost column have already become idiomatic, they can be difficult to read and cumbersome.

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

Exactly the same reason was behind adding contains member function to (unordered) associative containers in C++20.

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

- “How to find if a given key exists in a std::map”[3]
- “Determine if map contains a value for a key?”[4]

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

```cpp
if (m.find(key) != m.end()) {
    // m contains pair with key equal to given
}
```

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

```cpp
if (m.count(key)) {
    // m contains pair with key equal to given
}
```

The Proposal “Checking for Existence of an Element in Associative Containers”[5] 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:

<table>
<thead>
<tr>
<th>Prior C++20</th>
<th>Since C++20</th>
</tr>
</thead>
</table>
| if (m.find(key) != m.end()) {  
  // m contains pair with key equal to given  
} | if (m.contains(key)) {  
  // m contains pair with key equal to given  
} |
| if (m.count(key)) {  
  // m contains pair with key equal to given  
} | |

### 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[6] and last[7] elements, respectively.
- Similarly to Java, the BTreeSet structure in Rust also has `first()` and `last()` methods to get the first[8] and last[9] 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 keys order used by associative containers (`std::map`, `std::multimap`, `std::set`, `std::multiset`).

#### 3.2 Naming scheme

Especial 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[6,7] and Rust[8,9], 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 5 sequence containers in C++ STL:

- `std::array`
- `std::deque`
- `std::forward_list`
- `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 N4981, assuming the third naming scheme (`front/back`) is used.

#### 5.1 Associative container requirements

Add to section § 24.2.7.1 [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:

- § 24.4.4.1 [map.overview]
- § 24.4.5.1 [multimap.overview]
- § 24.4.6.1 [set.overview]
- § 24.4.7.1 [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:

- § 24.6.9.2 [flat.map.defn]
- § 24.6.10.2 [flat.multimap.defn]
- § 24.6.11.2 [flat.set.defn]
- § 24.6.12.2 [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:
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. How to find if a given key exists in a std::map
4. Determine if map contains a value for a key?

Proposal:

5. Checking for Existence of an Element in Associative Containers

Java:

6. SortedSet.first
7. SortedSet.last

Rust:

8. BTreeSet.first
9. BTreeSet.last