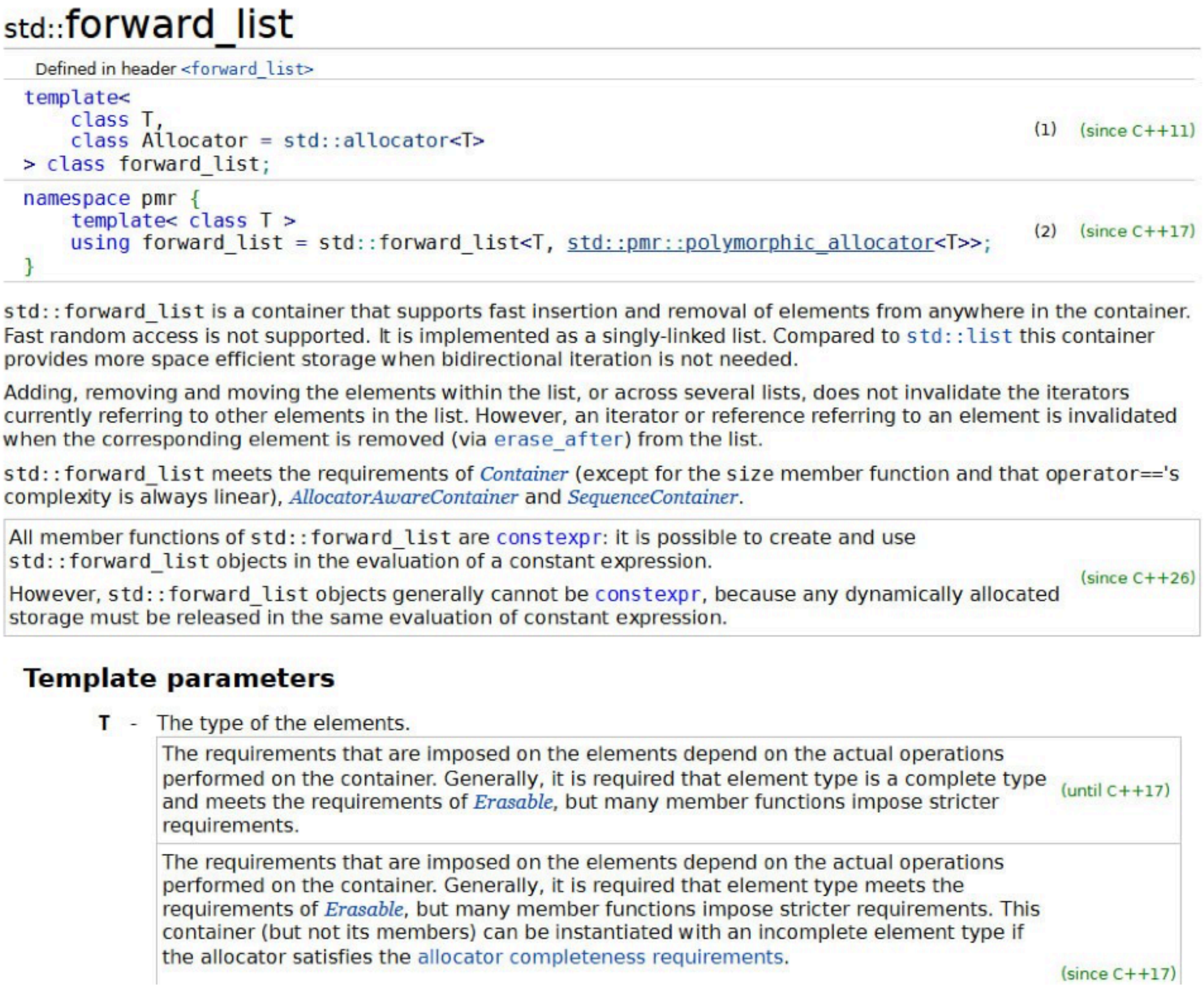
Задание 1: Итераторы
Ответ на вопрос: Контейнер std::forward_list использует однопроходные forward итераторы, потому что std::forward_list реализован как односвязный список, который позволяет перемещаться только в одном направлении от начала к концу.
Ответ на задание:



std::forward_list

Defined in header <forward_list>

```
template<
    class T,
    class Allocator = std::allocator<T>
> class forward_list;
```

(1) (since C++11)

```
namespace pmr {
    template< class T >
    using forward_list = std::forward_list<T, std::pmr::polymorphic_allocator<T>>;
}
```

(2) (since C++17)

std::forward_list is a container that supports fast insertion and removal of elements from anywhere in the container. Fast random access is not supported. It is implemented as a singly-linked list. Compared to std::list this container provides more space efficient storage when bidirectional iteration is not needed.

Adding, removing and moving the elements within the list, or across several lists, does not invalidate the iterators currently referring to other elements in the list. However, an iterator or reference referring to an element is invalidated when the corresponding element is removed (via erase_after) from the list.

std::forward_list meets the requirements of *Container* (except for the size member function and that operator=='s complexity is always linear), *AllocatorAwareContainer* and *SequenceContainer*.

All member functions of std::forward_list are constexpr: it is possible to create and use std::forward_list objects in the evaluation of a constant expression.

However, std::forward_list objects generally cannot be constexpr, because any dynamically allocated storage must be released in the same evaluation of constant expression.

(since C++26)

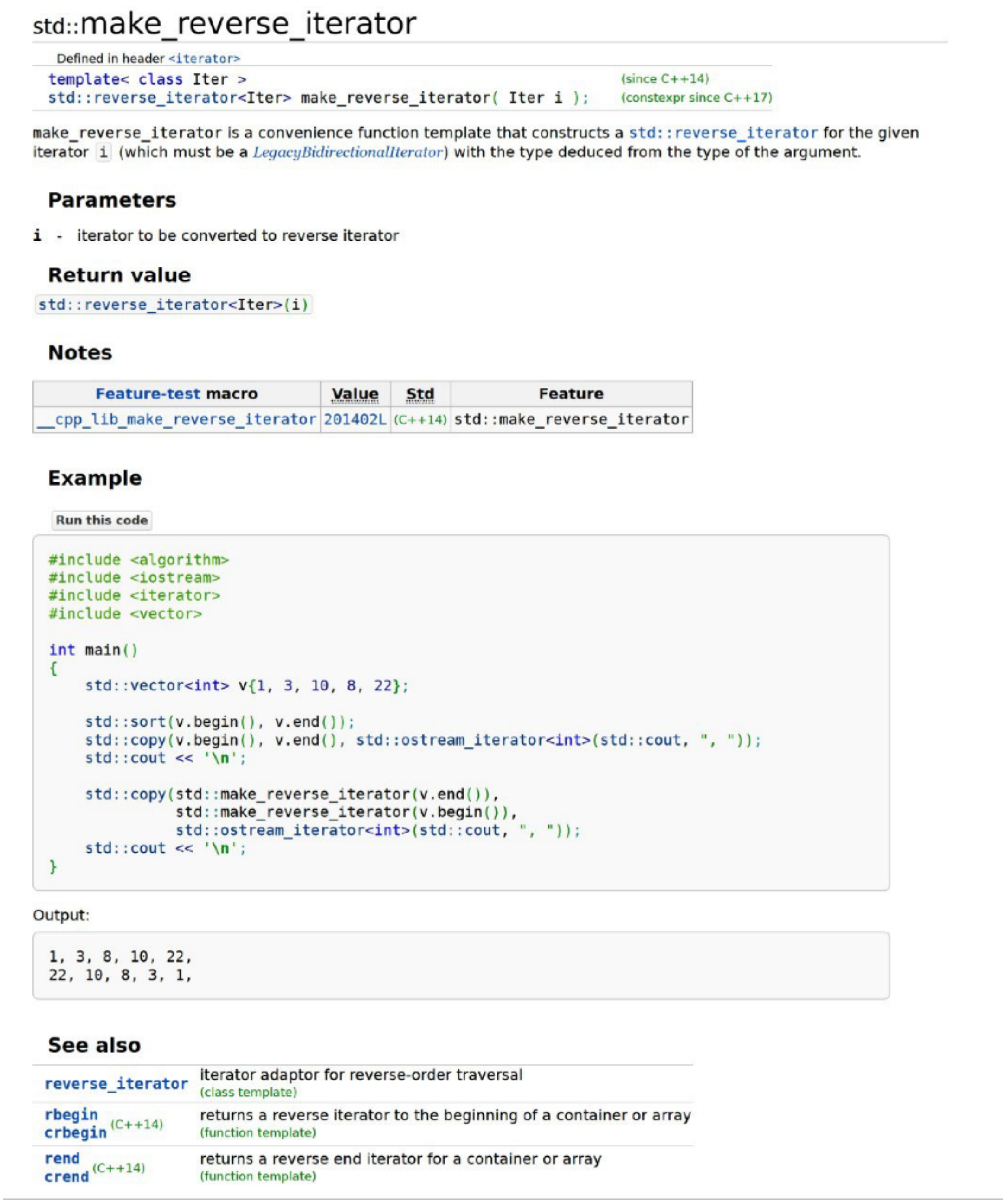
Template parameters

T - The type of the elements.

The requirements that are imposed on the elements depend on the actual operations performed on the container. Generally, it is required that element type is a complete type and meets the requirements of *Erasable*, but many member functions impose stricter requirements. (until C++17)

The requirements that are imposed on the elements depend on the actual operations performed on the container. Generally, it is required that element type meets the requirements of *Erasable*, but many member functions impose stricter requirements. This container (but not its members) can be instantiated with an incomplete element type if the allocator satisfies the allocator completeness requirements. (since C++17)

Задание 2: Работа с итераторами
Ответ на вопрос: Для преобразования обычного итератора в обратный можно использовать функцию std::make_reverse_iterator. Она создает std::reverse_iterator, который позволяет обходить контейнер в обратном порядке.
Ответ на задание:



std::make_reverse_iterator

Defined in header `<iterator>`

```
template< class Iter >                                        (since C++14)
std::reverse_iterator<Iter> make_reverse_iterator( Iter i );  (constexpr since C++17)
```

`make_reverse_iterator` is a convenience function template that constructs a `std::reverse_iterator` for the given iterator `i` (which must be a *LegacyBidirectionalIterator*) with the type deduced from the type of the argument.

Parameters

i - iterator to be converted to reverse iterator

Return value

`std::reverse_iterator<Iter>(i)`

Notes

Feature-test macro	Value	Std	Feature
`__cpp_lib_make_reverse_iterator`	201402L	(C++14)	`std::make_reverse_iterator`

Example

Run this code

```
#include <algorithm>
#include <iostream>
#include <iterator>
#include <vector>

int main()
{
    std::vector<int> v{1, 3, 10, 8, 22};

    std::sort(v.begin(), v.end());
    std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, ", "));
    std::cout << '\n';

    std::copy(std::make_reverse_iterator(v.end()),
              std::make_reverse_iterator(v.begin()),
              std::ostream_iterator<int>(std::cout, ", "));
    std::cout << '\n';
}
```

Output:

```
1, 3, 8, 10, 22,
22, 10, 8, 3, 1,
```

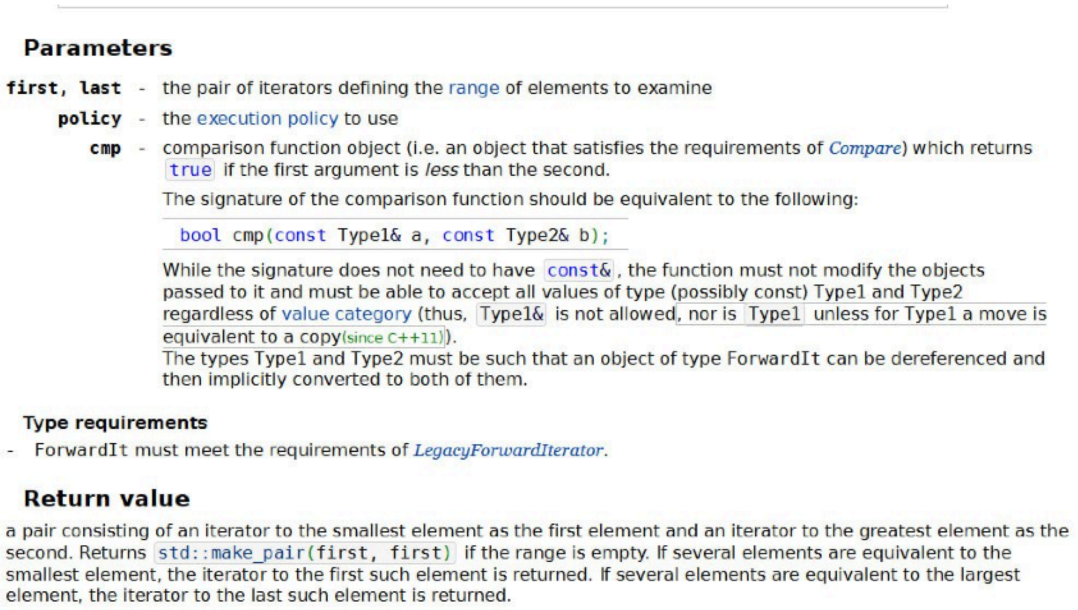
See also

`reverse_iterator`	iterator adaptor for reverse-order traversal (class template)
`rbegin` `crbegin` (C++14)	returns a reverse iterator to the beginning of a container or array (function template)
`rend` `crend` (C++14)	returns a reverse end iterator for a container or array (function template)

Задание 3: Алгоритмы STL

Ответ на вопрос: Для одновременного поиска минимального и максимального элементов в контейнере используется алгоритм std::minmax_element. Он возвращает пару итераторов, один из которых указывает на минимальный элемент, второй - на максимальный.

Ответ на задание:



Parameters

first, last - the pair of iterators defining the range of elements to examine

policy - the execution policy to use

cmp - comparison function object (i.e. an object that satisfies the requirements of *Compare*) which returns `true` if the first argument is *less* than the second.

The signature of the comparison function should be equivalent to the following:

```
bool cmp(const Type1& a, const Type2& b);
```

While the signature does not need to have `const&`, the function must not modify the objects passed to it and must be able to accept all values of type (possibly const) Type1 and Type2 regardless of value category (thus, `Type1&` is not allowed, nor is `Type1` unless for Type1 a move is equivalent to a copy(since C++11)).
The types Type1 and Type2 must be such that an object of type ForwardIt can be dereferenced and then implicitly converted to both of them.

Type requirements

- ForwardIt must meet the requirements of *LegacyForwardIterator*.

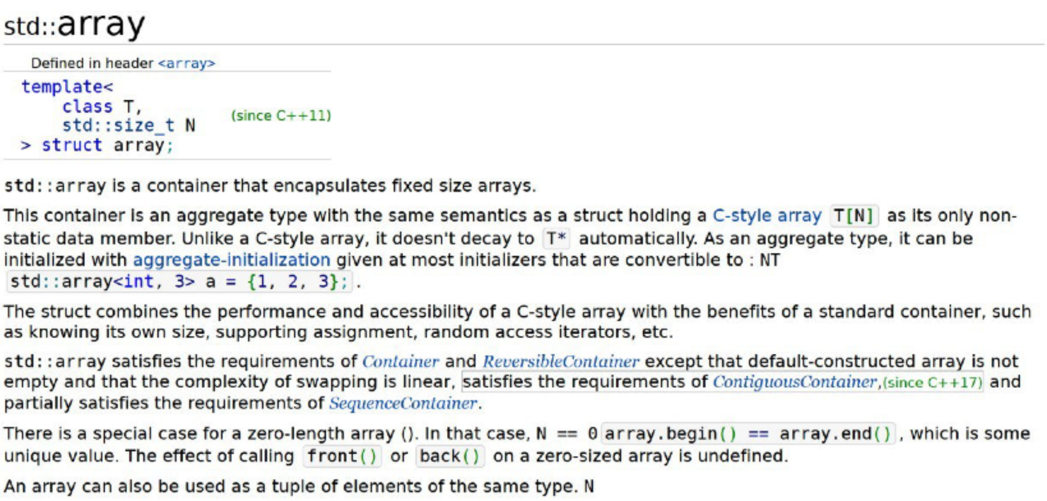
Return value

a pair consisting of an iterator to the smallest element as the first element and an iterator to the greatest element as the second. Returns `std::make_pair(first, first)` if the range is empty. If several elements are equivalent to the smallest element, the iterator to the first such element is returned. If several elements are equivalent to the largest element, the iterator to the last such element is returned.

Задание 4: Контейнеры STL

Ответ на вопрос: Основное отличие между std::array и std::vector заключается в том, что std::array имеет фиксированный размер, определяемый во время компиляции, тогда как std::vector может изменять свой размер во время выполнения программы. Это означает, что std::array не поддерживает динамическое изменение размера, но может быть более эффективным по памяти и производительности в случаях, когда размер известен заранее. С другой стороны, std::vector предоставляет гибкость за счет динамического управления памятью, что может сопровождаться дополнительными накладными расходами.

Ответ на задание:



std::array

Defined in header `<array>`

```
template<
    class T,
    std::size_t N
> struct array;
```
(since C++11)

`std::array` is a container that encapsulates fixed size arrays.

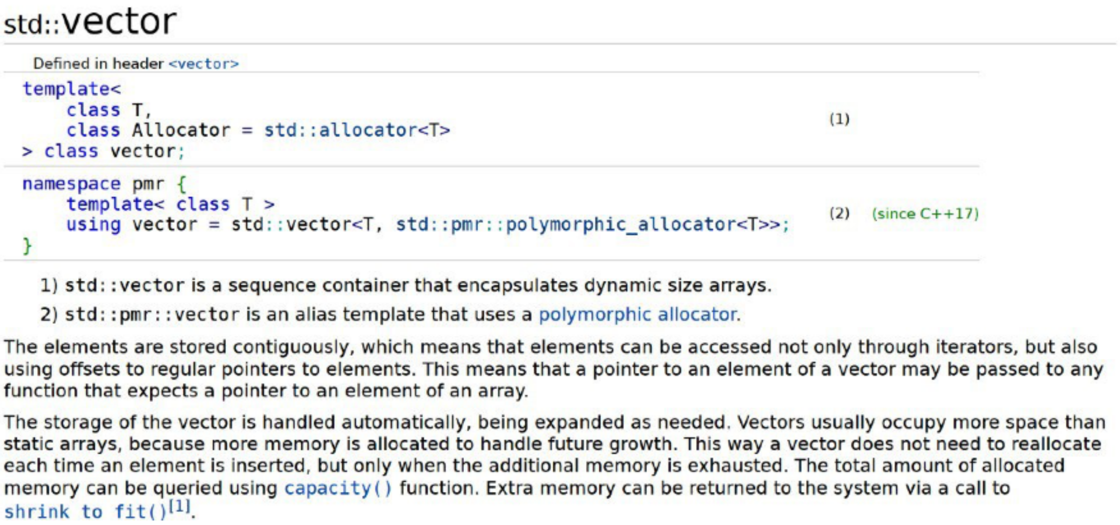
This container is an aggregate type with the same semantics as a struct holding a C-style array `T[N]` as its only non-static data member. Unlike a C-style array, it doesn't decay to `T*` automatically. As an aggregate type, it can be initialized with aggregate-initialization given at most initializers that are convertible to : NT
`std::array<int, 3> a = {1, 2, 3};`.

The struct combines the performance and accessibility of a C-style array with the benefits of a standard container, such as knowing its own size, supporting assignment, random access iterators, etc.

`std::array` satisfies the requirements of *Container* and *ReversibleContainer* except that default-constructed array is not empty and that the complexity of swapping is linear, satisfies the requirements of *ContiguousContainer*,(since C++17) and partially satisfies the requirements of *SequenceContainer*.

There is a special case for a zero-length array (). In that case, N == 0 `array.begin() == array.end()`, which is some unique value. The effect of calling `front()` or `back()` on a zero-sized array is undefined.

An array can also be used as a tuple of elements of the same type. N



std::vector

Defined in header `<vector>`

```
template<
    class T,
    class Allocator = std::allocator<T>
> class vector;
```
(1)

```
namespace pmr {
    template< class T >
    using vector = std::vector<T, std::pmr::polymorphic_allocator<T>>;
}
```
(2) (since C++17)

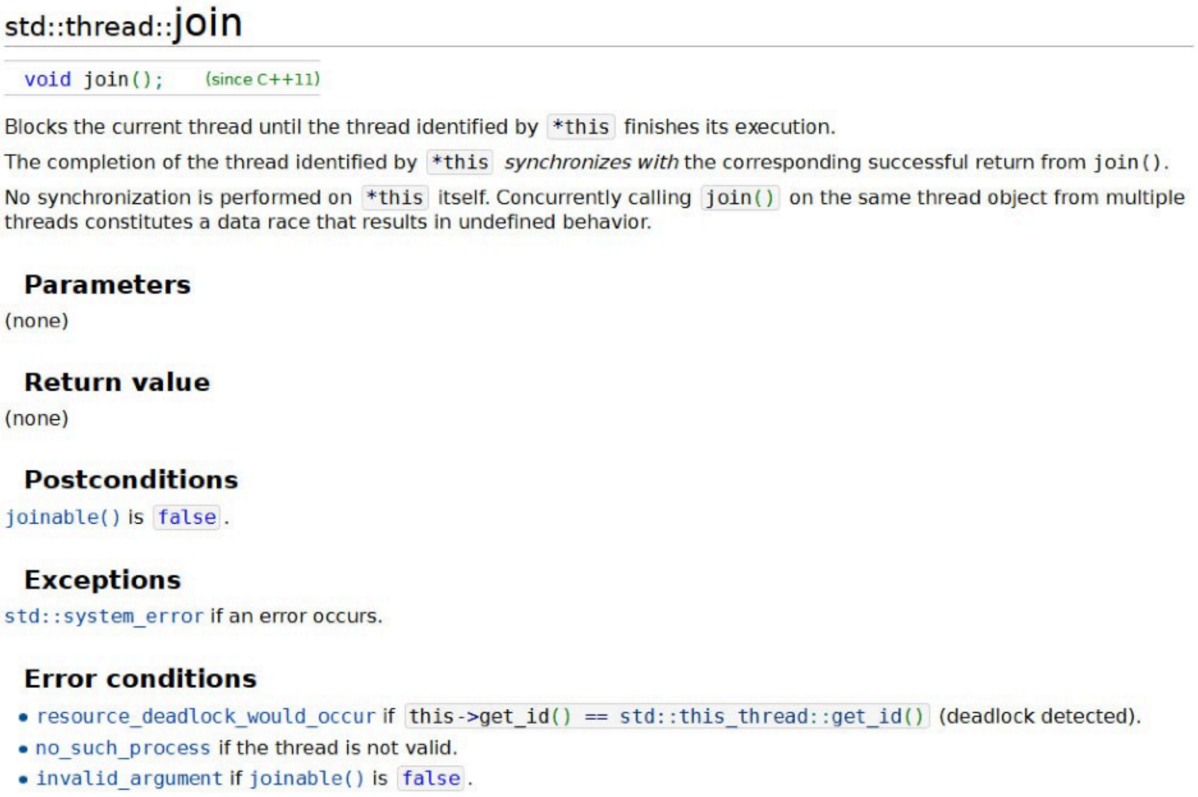
1) `std::vector` is a sequence container that encapsulates dynamic size arrays.

2) `std::pmr::vector` is an alias template that uses a polymorphic allocator.

The elements are stored contiguously, which means that elements can be accessed not only through iterators, but also using offsets to regular pointers to elements. This means that a pointer to an element of a vector may be passed to any function that expects a pointer to an element of an array.

The storage of the vector is handled automatically, being expanded as needed. Vectors usually occupy more space than static arrays, because more memory is allocated to handle future growth. This way a vector does not need to reallocate each time an element is inserted, but only when the additional memory is exhausted. The total amount of allocated memory can be queried using `capacity()` function. Extra memory can be returned to the system via a call to `shrink_to_fit()`[1].

Задание 5: Многопоточность в C++
Ответ на вопрос: Метод join() класса std::thred блокирует выполнение текущего потока до завершения потока, для которого он вызван. Это обеспечивает синхронизацию между потоками, гарантируя, что один поток завершится перед продолжением другого.
Ответ на задание:



std::thread::join

`void join();` (since C++11)

Blocks the current thread until the thread identified by `*this` finishes its execution.

The completion of the thread identified by `*this` *synchronizes with* the corresponding successful return from `join()`.

No synchronization is performed on `*this` itself. Concurrently calling `join()` on the same thread object from multiple threads constitutes a data race that results in undefined behavior.

Parameters

(none)

Return value

(none)

Postconditions

`joinable()` is `false`.

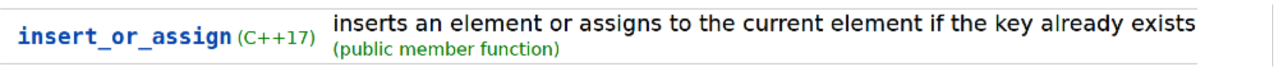
Exceptions

`std::system_error` if an error occurs.

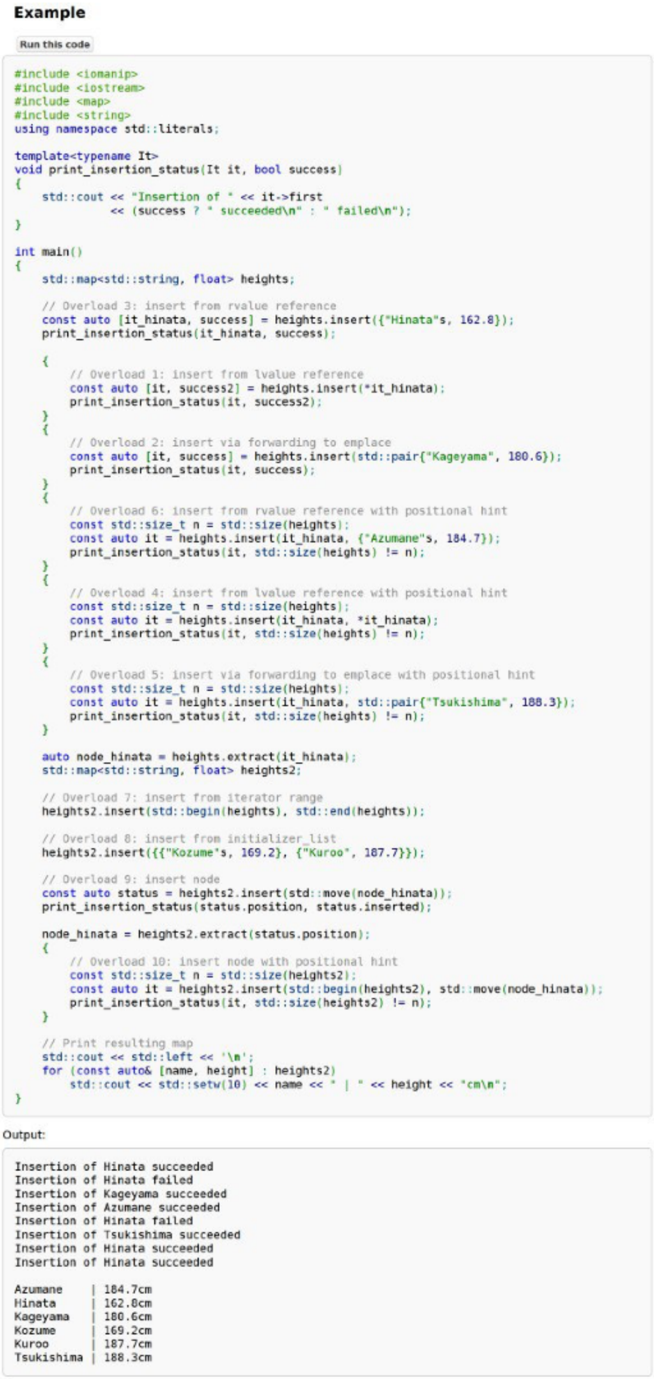
Error conditions

- `resource_deadlock_would_occur` if `this->get_id() == std::this_thread::get_id()` (deadlock detected).
- `no_such_process` if the thread is not valid.
- `invalid_argument` if `joinable()` is `false`.

Задание 6: Работа с контейнерами
Ответ на вопрос: Основное отличие между std::array и std::vector заключается в том, что std::array имеет фиксированный размер, определяемый во время компиляции, тогда как std::vector может изменять свой размер во время выполнения программы. Это означает, что std::array не поддерживает динамическое изменение размера, но может быть более эффективным по памяти и производительности в случаях, когда размер известен заранее. С другой стороны, std::vector предоставляет гибкость за счет динамического управления памятью, что может сопровождаться дополнительными накладными расходами
Ответ на задание:



insert_or_assign (C++17)	inserts an element or assigns to the current element if the key already exists (public member function)



Example

Run this code

```
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
using namespace std::literals;

template<typename It>
void print_insertion_status(It it, bool success)
{
    std::cout << "Insertion of " << it->first
              << (success ? " succeeded\n" : " failed\n");
}

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

    // Overload 3: insert from rvalue reference
    const auto [it_hinata, success] = heights.insert({"Hinata"s, 162.8});
    print_insertion_status(it_hinata, success);

    {
        // Overload 1: insert from lvalue reference
        const auto [it, success2] = heights.insert(*it_hinata);
        print_insertion_status(it, success2);
    }
    {
        // Overload 2: insert via forwarding to emplace
        const auto [it, success] = heights.insert(std::pair{"Kageyama", 180.6});
        print_insertion_status(it, success);
    }
    {
        // Overload 6: insert from rvalue reference with positional hint
        const std::size_t n = std::size(heights);
        const auto it = heights.insert(it_hinata, {"Azumane"s, 184.7});
        print_insertion_status(it, std::size(heights) != n);
    }
    {
        // Overload 4: insert from lvalue reference with positional hint
        const std::size_t n = std::size(heights);
        const auto it = heights.insert(it_hinata, *it_hinata);
        print_insertion_status(it, std::size(heights) != n);
    }
    {
        // Overload 5: insert via forwarding to emplace with positional hint
        const std::size_t n = std::size(heights);
        const auto it = heights.insert(it_hinata, std::pair{"Tsukishima", 188.3});
        print_insertion_status(it, std::size(heights) != n);
    }

    auto node_hinata = heights.extract(it_hinata);
    std::map<std::string, float> heights2;

    // Overload 7: insert from iterator range
    heights2.insert(std::begin(heights), std::end(heights));

    // Overload 8: insert from initializer_list
    heights2.insert({{"Kozume"s, 169.2}, {"Kuroo", 187.7}});

    // Overload 9: insert node
    const auto status = heights2.insert(std::move(node_hinata));
    print_insertion_status(status.position, status.inserted);

    node_hinata = heights2.extract(status.position);
    {
        // Overload 10: insert node with positional hint
        const std::size_t n = std::size(heights2);
        const auto it = heights2.insert(std::begin(heights2), std::move(node_hinata));
        print_insertion_status(it, std::size(heights2) != n);
    }

    // Print resulting map
    std::cout << std::left << '\n';
    for (const auto& [name, height] : heights2)
        std::cout << std::setw(10) << name << " | " << height << "cm\n";
}
```

Output:

```
Insertion of Hinata succeeded
Insertion of Hinata failed
Insertion of Kageyama succeeded
Insertion of Azumane succeeded
Insertion of Hinata failed
Insertion of Tsukishima succeeded
Insertion of Hinata succeeded
Insertion of Hinata succeeded

Azumane    | 184.7cm
Hinata     | 162.8cm
Kageyama   | 180.6cm
Kozume     | 169.2cm
Kuroo      | 187.7cm
Tsukishima | 188.3cm
```