[1. Контейнеры 2](#_Toc136284032)

[Последовательные контейнеры 2](#_Toc136284033)

[Ассоциативные контейнеры 3](#_Toc136284034)

[Неупорядоченные ассоциативные контейнеры 4](#_Toc136284035)

[Адаптеры для контейнеров 4](#_Toc136284036)

[1.1. Последовательные контейнеры 4](#_Toc136284037)

[1.1.1. Array 4](#_Toc136284038)

[1.1.2. Vector 11](#_Toc136284039)

[1.1.3. Deque 25](#_Toc136284040)

[1.1.4. Forward\_list 40](#_Toc136284041)

[1.1.5. List 53](#_Toc136284042)

[1.2. Ассоциативные контейнеры 69](#_Toc136284043)

[1.2.1. Set 69](#_Toc136284044)

[**1.2.1.1.** **Set** 69](#_Toc136284045)

[**1.2.1.2.** **Multiset** 80](#_Toc136284046)

[1.2.2. Map 90](#_Toc136284047)

[**1.2.2.1.** **Map** 90](#_Toc136284048)

[**1.2.2.2.** **Multimap** 103](#_Toc136284049)

[1.3. Неупорядоченные ассоциативные контейнеры 115](#_Toc136284050)

[1.3.1. Unordered\_set 115](#_Toc136284052)

[**1.3.1.1.** **Unordered\_set** 115](#_Toc136284053)

[**1.3.1.2.** **Unordered\_multiset** 127](#_Toc136284054)

[1.3.2. Unordered\_map 140](#_Toc136284055)

[**1.3.2.1.** **Unordered\_map** 140](#_Toc136284056)

[**1.3.2.2.** **Unordered\_multimap** 154](#_Toc136284057)

[1.4. Адаптеры для контейнеров 167](#_Toc136284058)

[1.4.3. Stack 167](#_Toc136284059)

[1.4.4. Queue 171](#_Toc136284060)

[1.4.5. Priority\_queue 174](#_Toc136284061)

[Последовательные контейнеры 178](#_Toc136284062)

[Ассоциативные контейнеры 179](#_Toc136284063)

[Неупорядоченные ассоциативные контейнеры 180](#_Toc136284064)

[1.5. Последовательные контейнеры 181](#_Toc136284065)

[1.5.1. Array 181](#_Toc136284066)

[1.5.2. Vector 209](#_Toc136284067)

[1.5.3. Deque 238](#_Toc136284068)

[1.5.4. Forward\_list 266](#_Toc136284069)

[1.5.5. List 272](#_Toc136284070)

[1.6. Ассоциативные контейнеры 297](#_Toc136284071)

[1.6.1. Set 297](#_Toc136284072)

[1.6.2. Map 321](#_Toc136284073)

[1.7. Неупорядоченные ассоциативные контейнеры 344](#_Toc136284074)

[1.7.1. Unordered\_set 344](#_Toc136284075)

[1.7.2. Unordered\_map 373](#_Toc136284076)

# **Контейнеры**

Контейнер - это объект-хранилище, который содержит коллекцию других объектов (его элементов). Они реализованы в виде шаблонов классов, что обеспечивает большую гибкость в поддержке различных типов элементов.

Контейнер управляет пространством хранения для своих элементов и предоставляет функции-члены для доступа к ним, как непосредственно, так и через итераторы (объекты-ссылки со свойствами, аналогичными свойствам указателей).

Контейнеры воспроизводят структуры, очень распространенные в программировании: динамические массивы (vector), очереди (queue), стеки (stack), кучи (priority\_queue), связные списки (list), деревья (set), ассоциативные массивы (map)...

Многие контейнеры имеют несколько общих функций-членов и совместно используют функциональные возможности. Решение о том, какой тип контейнера использовать для конкретной задачи, как правило, зависит не только от функциональности, предоставленной контейнером, но и от эффективности некоторых его элементов (сложности). Это особенно верно для контейнеров последовательностей, которые предлагают различные компромиссы в сложности между вставкой / удалением элементов и доступом к ним.

stack, queue и priority\_queue реализованы как адаптеры контейнеров. Адаптеры контейнеров - это не полноценные классы контейнеров, а классы, которые предоставляют определенный интерфейс, основанный на объекте одного из классов контейнеров (например, deque или list), для обработки элементов. Базовый контейнер инкапсулирован таким образом, что его элементы доступны через функции-члены адаптера контейнеров независимо от используемого базового контейнера.

# *Последовательные контейнеры*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Members | | [<array>](#_Array) | [<vector>](#_Vector) | [<deque>](#_Deque) | [<forward\_list>](#_Forward_list) | [<list>](#_List) |
| iterators | begin | [begin](#_array::begin) | [begin](#_vector::begin) | [begin](#_deque::begin) | [begin](#_forward_list::begin)  [before\_begin](#_forward_list::before_begin) | [begin](#_list::begin) |
| end | [end](#_array::end) | [end](#_vector::end) | [end](#_deque::end) | [end](#_forward_list::end) | [end](#_list::end) |
| rbegin | [rbegin](#_array::rbegin) | [rbegin](#_vector::rbegin) | [rbegin](#_deque::rbegin) |  | [rbegin](#_list::rbegin) |
| rend | [rend](#_array::rend) | [rend](#_vector::rend) | [rend](#_deque::rend) |  | [rend](#_list::rend) |
| const iterators | cbegin | [cbegin](#_array::cbegin) | [cbegin](#_vector::cbegin) | [cbegin](#_deque::cbegin) | [cbegin](#_forward_list::cbegin)  [cbefore\_begin](#_forward_list::cbefore_begin) | [cbegin](#_list::cbegin) |
| cend | [cend](#_array::cend) | [cend](#_vector::cend) | [cend](#_deque::cend) | [cend](#_forward_list::cend) | [cend](#_list::cend) |
| crbegin | [crbegin](#_array::crbegin) | [crbegin](#_vector::crbegin) | [crbegin](#_deque::crbegin) |  | [crbegin](#_list::crbegin) |
| crend | [crend](#_array::crend) | [crend](#_vector::crend) | [crend](#_deque::crend) |  | [crend](#_list::crend) |
| capacity | size | [size](#_array::size) | [size](#_vector::size) | [size](#_deque::size) |  | [size](#_list::size) |
| max\_size | [max\_size](#_array::max_size) | [max\_size](#_vector::max_size) | [max\_size](#_deque::max_size) | [max\_size](#_forward_list::max_size) | [max\_size](#_list::max_size) |
| empty | [empty](#_array::empty) | [empty](#_vector::empty) | [empty](#_deque::empty) | [empty](#_forward_list::empty) | [empty](#_list::empty) |
| resize |  | [resize](#_vector::resize) | [resize](#_deque::resize) | [resize](#_forward_list::resize) | [resize](#_list::resize) |
| shrink\_to\_fit |  | [shrink\_to\_fit](#_vector::shrink_to_fit) | [shrink\_to\_fit](#_deque::shrink_to_fit) |  |  |
| capacity |  | [capacity](#_vector::capacity) |  |  |  |
| reserve |  | [reserve](#_vector::reserve) |  |  |  |
| element access | front | [front](#_array::front) | [front](#_vector::front) | [front](#_deque::front) | [front](#_forward_list::front) | [front](#_list::front) |
| back | [back](#_array::back) | [back](#_vector::back) | [back](#_deque::back) |  | [back](#_list::back) |
| operator[] | [operator[]](#_array::operator[]) | [operator[]](#_vector::operator[]) | [operator[]](#_deque::operator[]) |  |  |
| at | [at](#_array::at) | [at](#_vector::at) | [at](#_deque::at) |  |  |
| modifiers | assign |  | [assign](#_vector::assign) | [assign](#_deque::assign) | [assign](#_forward_list::assign) | [assign](#_list::assign) |
| emplace |  | [emplace](#_vector::emplace) | [emplace](#_deque::emplace) | [emplace\_after](#_forward_list::emplace_after) | [emplace](#_list::emplace) |
| insert |  | [insert](#_vector::insert) | [insert](#_deque::insert) | [insert\_after](#_forward_list::insert_after) | [insert](#_list::insert) |
| erase |  | [erase](#_vector::erase) | [erase](#_deque::erase) | [erase\_after](#_forward_list::erase_after) | [erase](#_list::erase) |
| emplace\_back |  | [emplace\_back](#_vector::emplace_back) | [emplace\_back](#_deque::emplace_back) |  | [emplace\_back](#_list::emplace_back) |
| push\_back |  | [push\_back](#_vector::push_back) | [push\_back](#_deque::push_back) |  | [push\_back](#_list::push_back) |
| pop\_back |  | [pop\_back](#_vector::pop_back) | [pop\_back](#_deque::pop_back) |  | [pop\_back](#_list::pop_back) |
| emplace\_front |  |  | [emplace\_front](#_deque::emplace_front) | [emplace\_front](#_forward_list::emplace_front) | [emplace\_front](#_list::emplace_front) |
| push\_front |  |  | [push\_front](#_deque::push_front) | [push\_front](#_forward_list::push_front) | [push\_front](#_list::push_front) |
| pop\_front |  |  | [pop\_front](#_deque::pop_front) | [pop\_front](#_forward_list::pop_front) | [pop\_front](#_list::pop_front) |
| clear |  | [clear](#_vector::clear) | [clear](#_deque::clear) | [clear](#_forward_list::clear) | [clear](#_list::clear) |
| swap | [swap](#_array::swap) | [swap](#_vector::swap) | [swap](#_deque::swap) | [swap](#_forward_list::swap) | [swap](#_list::swap) |
| list operations | splice |  |  |  | [splice\_after](#_forward_list::splice_after) | [splice](#_list::splice) |
| remove |  |  |  | [remove](#_forward_list::remove) | [remove](#_list::remove) |
| remove\_if |  |  |  | [remove\_if](#_forward_list::remove_if) | [remove\_if](#_list::remove_if) |
| unique |  |  |  | [unique](#_forward_list::unique) | [unique](#_list::unique) |
| merge |  |  |  | [merge](#_forward_list::merge) | [merge](#_list::merge) |
| sort |  |  |  | [sort](#_forward_list::sort) | [sort](#_list::sort) |
| reserve |  |  |  | [reserve](#_forward_list::reverse) | [reserve](#_list::reverse) |
| observers | get\_allocator |  | [get\_allocator](#_vector::get_allocator) | [get\_allocator](#_deque::get_allocator) | [get\_allocator](#_forward_list::get_allocator) | [get\_allocator](#_list::ger_allocator) |
| data | [data](#_array::data) | [data](#_vector::data) |  |  |  |

# *Ассоциативные контейнеры*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Members | | <set> | | <map> | |
| [set](#_Set) | [multiset](#_Multiset) | [map](#_Map) | [multimap](#_Multimap) |
| iterators | begin | [begin](#_set::begin) | [begin](#_multiset::begin) | [begin](#_map::begin) | [begin](#_multimap::begin) |
| end | [end](#_set::end) | [end](#_multiset::end) | [end](#_map::end) | [end](#_multimap::end) |
| rbegin | [rbegin](#_set::rbegin) | [rbegin](#_multiset::rbegin) | [rbegin](#_map::rbegin) | [rbegin](#_multimap::rbegin) |
| rend | [rend](#_set::rend) | [rend](#_multiset::rend) | [rend](#_map::rend) | [rend](#_multimap::rend) |
| const iterators | cbegin | [cbegin](#_set::cbegin) | [cbegin](#_multiset::cbegin) | [cbegin](#_map::cbegin) | [cbegin](#_multimap::cbegin) |
| cend | [cend](#_set::cend) | [cend](#_multiset::cend) | [cend](#_map::cend) | [cend](#_multimap::cend) |
| crbegin | [crbegin](#_set::crbegin) | [crbegin](#_multiset::crbegin) | [crbegin](#_map::crbegin) | [crbegin](#_multimap::crbegin) |
| crend | [crend](#_set::crend) | [crend](#_multiset::crend) | [crend](#_map::crend) | [crend](#_multimap::crend) |
| capacity | size | [size](#_set::size) | [size](#_multiset::size) | [size](#_map::size) | [size](#_multimap::size) |
| max\_size | [max\_size](#_set::max_size) | [max\_size](#_multiset::max_size) | [max\_size](#_map::max_size) | [max\_size](#_multimap::max_size) |
| empty | [empty](#_set::empty) | [empty](#_multiset::empty) | [empty](#_map::empty) | [empty](#_multimap::empty) |
| reserve |  |  |  |  |
| element access | at |  |  | [at](#_map::at) |  |
| operator[] |  |  | [operator[]](#_map::operator[]) |  |
| modifiers | emplace | [emplace](#_set::emplace) | [emplace](#_multiset::emplace) | [emplace](#_map::emplace) | [emplace](#_multimap::emplace) |
| emplace\_hint | [emplace\_hint](#_set::emplace_hint) | [emplace\_hint](#_multiset::emplace_hint) | [emplace\_hint](#_map::emplace_hint) | [emplace\_hint](#_multimap::emplace_hint) |
| insert | [insert](#_set::insert) | [insert](#_multiset::insert) | [insert](#_map::insert) | [insert](#_multimap::insert) |
| erase | [erase](#_set::erase) | [erase](#_multiset::erase) | [erase](#_map::erase) | [erase](#_multimap::erase) |
| clear | [clear](#_set::clear) | [clear](#_multiset::clear) | [clear](#_map::clear) | [clear](#_multimap::clear) |
| swap | [swap](#_set::swap) | [swap](#_multiset::swap) | [swap](#_map::swap) | [swap](#_multimap::swap) |
| operations | count | [count](#_set::count) | [count](#_multiset::count) | [count](#_map::count) | [count](#_multimap::count) |
| find | [find](#_set::find) | [find](#_multiset::find) | [find](#_map::find) | [find](#_multimap::find) |
| equal\_range | [equal\_range](#_set::equal_range) | [equal\_range](#_multiset::equal_range) | [equal\_range](#_map::equal_range) | [equal\_range](#_multimap::equal_range) |
| lower\_bound | [lower\_bound](#_set::lower_bound) | [lower\_bound](#_multiset::lower_bound) | [lower\_bound](#_map::lower_bound) | [lower\_bound](#_multimap::lower_bound) |
| upper\_bound | [upper\_bound](#_set::upper_bound) | [upper\_bound](#_multiset::upper_bound) | [upper\_bound](#_map::upper_bound) | [upper\_bound](#_multimap::upper_bound) |
| observers | get\_allocator | [get\_allocator](#_set::get_allocator) | [get\_allocator](#_multiset::get_allocator) | [get\_allocator](#_map::get_allocator) | [get\_allocator](#_multimap::get_allocator) |
| key\_comp | [key\_comp](#_set::key_comp) | [key\_comp](#_multiset::key_comp) | [key\_comp](#_map::key_comp) | [key\_comp](#_multimap::key_comp) |
| value\_comp | [value\_comp](#_set::value_comp) | [value\_comp](#_multiset::value_comp) | [value\_comp](#_map::value_comp) | [value\_comp](#_multimap::value_comp) |
| key\_eq |  |  |  |  |
| hash\_function |  |  |  |  |
| buckets | bucket |  |  |  |  |
| bucket\_count |  |  |  |  |
| bucket\_size |  |  |  |  |
| max\_bucket\_count |  |  |  |  |
| hash policy | rehash |  |  |  |  |
| load\_factor |  |  |  |  |
| max\_load\_factor |  |  |  |  |

# *Неупорядоченные ассоциативные контейнеры*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Members | | <unordered\_set> | | <unordered\_map> | |
| [unordered\_set](#_Unordered_set) | [unordered\_multiset](#_Unordered_multiset) | [unordered\_map](#_Unordered_map) | [unordered\_multimap](#_Unordered_multimap) |
| iterators | begin | [begin](#_unordered_set::begin) | [begin](#_unordered_multiset::begin) | [begin](#_unordered_map::begin) | [begin](#_unordered_multimap::begin) |
| end | [end](#_unordered_set::end) | [end](#_unordered_multiset::end) | [end](#_unordered_map::end) | [end](#_unordered_multimap::end) |
| rbegin |  |  |  |  |
| rend |  |  |  |  |
| const iterators | cbegin | [cbegin](#_unordered_set::cbegin) | [cbegin](#_unordered_multiset::cbegin) | [cbegin](#_unordered_map::cbegin) | [cbegin](#_unordered_multimap::cbegin) |
| cend | [cend](#_unordered_set::cend) | [cend](#_unordered_multiset::cend) | [cend](#_unordered_map::cend) | [cend](#_unordered_multimap::cend) |
| crbegin |  |  |  |  |
| crend |  |  |  |  |
| capacity | size | [size](#_unordered_set::size) | [size](#_unordered_multiset::size) | [size](#_unordered_map::size) | [size](#_unordered_multimap::size) |
| max\_size | [max\_size](#_unordered_set::max_size) | [max\_size](#_unordered_multiset::max_size) | [max\_size](#_unordered_map::max_size) | [max\_size](#_unordered_multimap::max_size) |
| empty | [empty](#_unordered_set::empty) | [empty](#_unordered_multiset::empty) | [empty](#_unordered_map::empty) | [empty](#_unordered_multimap::empty) |
| reserve | [reserve](#_unordered_set::reserve) | [reserve](#_unordered_multiset::reserve) | [reserve](#_unordered_map::reserve) | [reserve](#_unordered_multimap::reserve) |
| element access | at |  |  | [at](#_unordered_map::at) |  |
| operator[] |  |  | [operator[]](#_unordered_map::operator[]) |  |
| modifiers | emplace | [emplace](#_unordered_set::emplace) | [emplace](#_unordered_multiset::emplace) | [emplace](#_unordered_map::emplace) | [emplace](#_unordered_multimap::emplace) |
| emplace\_hint | [emplace\_hint](#_unordered_set::emplace_hint) | [emplace\_hint](#_unordered_multiset::emplace_hint) | [emplace\_hint](#_unordered_map::emplace_hint) | [emplace\_hint](#_unordered_multimap::emplace_hint) |
| insert | [insert](#_unordered_set::insert) | [insert](#_unordered_multiset::insert) | [insert](#_unordered_map::insert) | [insert](#_unordered_multimap::insert) |
| erase | [erase](#_unordered_set::erase) | [erase](#_unordered_multiset::erase) | [erase](#_unordered_map::erase) | [erase](#_unordered_multimap::erase) |
| clear | [clear](#_unordered_set::clear) | [clear](#_unordered_multiset::clear) | [clear](#_unordered_map::clear) | [clear](#_unordered_multimap::clear) |
| swap | [swap](#_unordered_set::swap) | [swap](#_unordered_multiset::swap) | [swap](#_unordered_map::swap) | [swap](#_unordered_multimap::swap) |
| operations | count | [count](#_unordered_set::count) | [count](#_unordered_multiset::count) | [count](#_unordered_map::count) | [count](#_unordered_multimap::count) |
| find | [find](#_unordered_set::find) | [find](#_unordered_multiset::find) | [find](#_unordered_map::find) | [find](#_unordered_multimap::find) |
| equal\_range | [equal\_range](#_unordered_set::equal_range) | [equal\_range](#_unordered_multiset::equal_range) | [equal\_range](#_unordered_map::equal_range) | [equal\_range](#_unordered_multimap::equal_range) |
| lower\_bound |  |  |  |  |
| upper\_bound |  |  |  |  |
| observers | get\_allocator | [get\_allocator](#_unordered_set::get_allocator) | [get\_allocator](#_unordered_multiset::get_allocator) | [get\_allocator](#_unordered_map::get_allocator) | [get\_allocator](#_unordered_multimap::get_allocator) |
| key\_comp |  |  |  |  |
| value\_comp |  |  |  |  |
| key\_eq | [key\_eq](#_unordered_set::key_eq) | [key\_eq](#_unordered_multiset::key_eq) | [key\_eq](#_unordered_map::key_eq) | [key\_eq](#_unordered_multimap::key_eq) |
| hash\_function | [hash\_function](#_unordered_set::hash_function) | [hash\_function](#_unordered_multiset::hash_function) | [hash\_function](#_unordered_map::hash_function) | [hash\_function](#_unordered_multimap::hash_function) |
| buckets | bucket | [bucket](#_unordered_set::bucket) | [bucket](#_unordered_multiset::bucket) | [bucket](#_unordered_map::bucket) | [bucket](#_unordered_multimap::bucket) |
| bucket\_count | [bucket\_count](#_unordered_set::bucket_count) | [bucket\_count](#_unordered_multiset::bucket_count) | [bucket\_count](#_unordered_map::bucket_count) | [bucket\_count](#_unordered_multimap::bucket_count) |
| bucket\_size | [bucket\_size](#_unordered_set::bucket_size) | [bucket\_size](#_unordered_multiset::bucket_size) | [bucket\_size](#_unordered_map::bucket_size) | [bucket\_size](#_unordered_multimap::bucket_size) |
| max\_bucket\_count | [max\_bucket\_count](#_unordered_set::max_bucket_count) | [max\_bucket\_count](#_unordered_multiset::max_bucket_coun) | [max\_bucket\_count](#_unordered_map::max_bucket_count) | [max\_bucket\_count](#_unordered_multimap::max_bucket_coun) |
| hash policy | rehash | [rehash](#_unordered_set::rehash) | [rehash](#_unordered_multiset::rehash) | [rehash](#_unordered_map::rehash) | [rehash](#_unordered_multimap::rehash) |
| load\_factor | [load\_factor](#_unordered_set::load_factor) | [load\_factor](#_unordered_multiset::load_factor) | [load\_factor](#_unordered_map::load_factor) | [load\_factor](#_unordered_multimap::load_factor) |
| max\_load\_factor | [max\_load\_factor](#_unordered_set::max_load_factor) | [max\_load\_factor](#_unordered_multiset::max_load_factor) | [max\_load\_factor](#_unordered_map::max_load_factor) | [max\_load\_factor](#_unordered_multimap::max_load_factor) |

# *Адаптеры для контейнеров*

|  |  |  |  |
| --- | --- | --- | --- |
| Members | [<stack>](#_Stack) | [<queue>](#_Queue) | [<priority\_queue>](#_Priority_queue) |
| empty | [empty](#_stack::empty) | [empty](#_queue::empty) | [empty](#_priority_queue::empty) |
| size | [size](#_stack::size) | [size](#_queue::size) | [size](#_priority_queue::size) |
| top | [top](#_stack::top) |  | [top](#_priority_queue::top) |
| push | [push](#_stack::push) | [push](#_queue::push) | [push](#_priority_queue::push) |
| emplace | [emplace](#_stack::emplace) | [emplace](#_queue::emplace) | [emplace](#_priority_queue::emplace) |
| pop | [pop](#_stack::pop) | [pop](#_queue::pop) | [pop](#_priority_queue::pop) |
| swap | [swap](#_stack::swap) | [swap](#_queue::swap) | [swap](#_priority_queue::swap) |
| front |  | front |  |
| back |  | back |  |

# **Последовательные контейнеры**

## **Array**

template < class T, size\_t N > класс array; // общий шаблон

Массивы - это контейнеры последовательностей фиксированного размера: они содержат определенное количество элементов, упорядоченных в строгой линейной последовательности.

Внутренне массив не хранит никаких данных, кроме элементов, которые он содержит (даже его размер, который является параметром шаблона, фиксируемым во время компиляции). Он эффективен по размеру хранилища, как и обычный массив, объявленный с использованием квадратных скобок ([]). Этот класс предоставляет набор методов и функций-членов, чтобы массивы можно было использовать как стандартные контейнеры.

В отличие от других стандартных контейнеров, массивы имеют фиксированный размер и не управляют выделением своих элементов через аллокатор: они являются агрегатным типом, инкапсулирующим массив элементов фиксированного размера. Поэтому их размер не может быть динамически изменен (в отличие, например, от контейнера vector).

Массивы нулевого размера допустимы, но не должны разыменовываться (методы front, back и data).

В отличие от других контейнеров в стандартной библиотеке, обмен двух массивов является линейной операцией, которая включает обмен всех элементов по отдельности, что обычно является менее эффективной операцией. С другой стороны, это позволяет итераторам элементов в обоих контейнерах сохранять их первоначальную связь с контейнерами.

Еще одной уникальной особенностью контейнера array является то, что их можно рассматривать как объекты tuple: заголовочный файл <array> перегружает функцию get для доступа к элементам массива, как если бы это был tuple, а также определяет специализированные типы tuple\_size и tuple\_element.

**Свойства контейнера:**

* *Последовательность*: элементы в последовательных контейнерах упорядочены в строгой линейной последовательности. Доступ к отдельным элементам осуществляется по их положению в этой последовательности.
* *Последовательное расположение в памяти*: элементы хранятся в памяти последовательно, что позволяет получать доступ к элементам за константное время. Указатели на элементы могут быть смещены для доступа к другим элементам.
* *Фиксированный размер*: контейнер использует неявные конструкторы и деструкторы для статического выделения необходимого пространства. Размер массива фиксирован во время компиляции и не меняется во время выполнения программы. Никаких затрат памяти или времени.

**Параметры шаблона**:

* T: Тип элементов, содержащихся в массиве.
* N: Размер массива, выраженный в количестве элементов.

#### **array::begin**

begin - возвращает итератор, указывающий на первый элемент контейнера.

Пример использования методов begin() и end() в std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 5> myarray = { 2, 16, 77, 34, 50 };

std::cout << "Array:";

for (auto it = myarray.begin(); it != myarray.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::end**

end - возвращает итератор, указывающий на элемент, следующий за последним элементом контейнера.

#### **array::rbegin**

rbegin - возвращает обратный итератор, указывающий на последний элемент контейнера.

Пример использования методов rbegin() и rend() в std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 4> myarray = { 4, 26, 80, 14 };

std::cout << "Array:";

for (auto rit = myarray.rbegin(); rit < myarray.rend(); ++rit)

std::cout << ' ' << \*rit;

std::cout << '\n';

return 0;

}

Обратите внимание, как обратный итератор выполняет итерацию по вектору обратным образом, увеличивая итератор.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::rend**

rend - возвращает обратный итератор, указывающий на элемент, предшествующий первому элементу контейнера.

#### **array::cbegin**

cbegin - возвращает константный итератор, указывающий на первый элемент контейнера.

Пример использования методов cbegin() и cend() для итерации по элементам std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 5> myarray = { 2, 16, 77, 34, 50 };

std::cout << "Array:";

for (auto it = myarray.cbegin(); it != myarray.cend(); ++it)

std::cout << ' ' << \*it; // не удается изменить \*it

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::cend**

cend - возвращает константный итератор, указывающий на элемент, следующий за последним элементом контейнера.

#### **array::crbegin**

crbegin - возвращает константный обратный итератор, указывающий на последний элемент контейнера.

Пример использования методов cbegin() и crend() в std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 6> myarray = { 10, 20, 30, 40, 50, 60 };

std::cout << "Array:";

for (auto rit = myarray.crbegin(); rit < myarray.crend(); ++rit)

std::cout << ' ' << \*rit; // не удается изменить \*rit

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::crend**

crend - возвращает константный обратный итератор, указывающий на элемент, предшествующий первому элементу контейнера.

#### **array::size**

size - возвращает количество элементов в контейнере.

Пример использования метода size() в std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 5> myints;

std::cout << "size of myints: " << myints.size() << std::endl;

std::cout << "sizeof(myints): " << sizeof(myints) << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::max\_size**

max\_size - возвращает максимально возможное количество элементов, которое может содержать контейнер.

Пример использования метода max\_size() в std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 10> myints;

std::cout << "size of myints: " << myints.size() << '\n';

std::cout << "max\_size of myints: " << myints.max\_size() << '\n';

return 0;

}

size and max\_size массива всегда совпадают.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::empty**

empty - проверяет, является ли контейнер пустым.

Пример использования метода empty() в std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 0> first;

std::array<int, 5> second;

std::cout << "first " << (first.empty() ? "is empty" : "is not empty") << '\n';

std::cout << "second " << (second.empty() ? "is empty" : "is not empty") << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::front**

front - возвращает ссылку на первый элемент контейнера.

Пример использования метода front() в std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 3> myarray = { 2, 16, 77 };

std::cout << "front is: " << myarray.front() << std::endl; // 2

std::cout << "back is: " << myarray.back() << std::endl; // 77

myarray.front() = 100;

std::cout << "myarray now contains:";

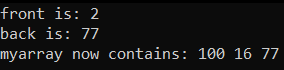
for (int& x : myarray) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::back**

back - возвращает ссылку на последний элемент контейнера.

Пример использования метода back() в std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 3> myarray = { 5, 19, 77 };

std::cout << "front is: " << myarray.front() << std::endl; // 5

std::cout << "back is: " << myarray.back() << std::endl; // 77

myarray.back() = 50;

std::cout << "myarray now contains:";

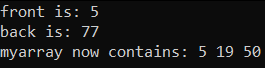
for (int& x : myarray) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::operator[]**

operator[] - обеспечивает доступ к элементам контейнера по индексу.

Пример использования метода operator[] в std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 10> myarray;

unsigned int i;

// присвоить некоторые значения

for (i = 0; i < 10; i++) myarray[i] = i;

// вывод

std::cout << "myarray contains:";

for (i = 0; i < 10; i++)

std::cout << ' ' << myarray[i];

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::at**

at - обеспечивает доступ к элементам контейнера по индексу с проверкой границ.

Пример использования метода at() в std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 10> myarray;

// присвоить некоторые значения:

for (int i = 0; i < 10; i++) myarray.at(i) = i + 1;

// вывод

std::cout << "myarray contains:";

for (int i = 0; i < 10; i++)

std::cout << ' ' << myarray.at(i);

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::swap**

swap - обменивает содержимое двух контейнеров.

Пример использования метода swap() в std::array:

#include <iostream>

#include <array>

int main() {

std::array<int, 5> first = { 10, 20, 30, 40, 50 };

std::array<int, 5> second = { 11, 22, 33, 44, 55 };

first.swap(second);

std::cout << "first:";

for (int& x : first) std::cout << ' ' << x;

std::cout << '\n';

std::cout << "second:";

for (int& x : second) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **array::data**

data - возвращает указатель на первый элемент контейнера.

Пример использования метода data() в std::array:

#include <iostream>

#include <cstring>

#include <array>

int main() {

const char\* cstr = "Test string";

std::array<char, 12> charray;

std::memcpy(charray.data(), cstr, 12);

std::cout << charray.data() << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

## **Vector**

template < class T, class Alloc = allocator<T> > класс vector; // общий шаблон

Векторы являются контейнерами-последовательностями, представляющими массивы, которые могут изменять свой размер.

Как и массивы, векторы используют последовательное расположение элементов в памяти, что означает, что к их элементам также можно получить доступ с использованием смещений в обычных указателях на элементы, и так же эффективно, как в массивах. Однако, в отличие от массивов, их размер может динамически изменяться, и их хранилище автоматически управляется контейнером.

Внутренне векторы используют динамически выделенный массив для хранения своих элементов. Этот массив может потребоваться перевыделить для увеличения размера, когда добавляются новые элементы, что означает выделение нового массива и перемещение всех элементов в него. Это относительно затратная задача по времени обработки, и поэтому векторы не перевыделяют память каждый раз, когда элемент добавляется в контейнер.

Вместо этого контейнеры векторов могут выделять некоторое дополнительное хранилище для возможного роста, и поэтому контейнер может иметь фактическую ёмкость, превышающую хранилище, необходимое для содержания его элементов (т.е. его размер). Библиотеки могут использовать различные стратегии роста для балансировки между использованием памяти и перевыделениями, но в любом случае, перевыделения должны происходить только при логарифмическом увеличении размера, чтобы вставка отдельных элементов в конец вектора могла быть выполнена с амортизированной константной сложностью времени (см. push\_back).

Таким образом, по сравнению с массивами векторы потребляют больше памяти в обмен на возможность динамического управления хранилищем и эффективного роста.

По сравнению с другими динамическими контейнерами-последовательностями (деками, списками и односвязными списками), векторы очень эффективны в доступе к своим элементам (как массивы) и относительно эффективны при добавлении или удалении элементов с конца. Для операций, включающих вставку или удаление элементов на позициях, от позиции, отличных от конца, они работают хуже других контейнеров и имеют менее последовательные итераторы и ссылки, чем списки и односвязные списки.

**Свойства контейнера**:

* *Последовательность*: элементы в контейнерах-последовательностях упорядочены в строгой линейной последовательности. Доступ к отдельным элементам осуществляется по их положению в этой последовательности.
* *Динамический массив*: позволяет прямой доступ к любому элементу в последовательности, даже с использованием арифметики указателей, и обеспечивает относительно быстрое добавление/удаление элементов в конце последовательности.
* *Ориентированный на распределитель*: контейнер использует объект аллокатора для динамической обработки своих потребностей в памяти.

**Параметры шаблона:**

* Шаблонный параметр T определяет тип элементов, которые будут храниться в vector. Это может быть любой тип данных, например, int, double или пользовательский тип.
* Alloc является необязательным параметром шаблона и представляет аллокатор, который используется для выделения памяти под элементы vector. По умолчанию используется аллокатор allocator<T>, который выделяет память с помощью оператора new. Однако пользователь может предоставить свою реализацию аллокатора, если требуется специфическое поведение при выделении и освобождении памяти.

#### **vector::begin**

begin - возвращает итератор, указывающий на первый элемент контейнера.

Пример использования методов begin() и end() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

for (int i = 1; i <= 5; i++) myvector.push\_back(i);

std::cout << "myvector contains:";

for (std::vector<int>::iterator it = myvector.begin(); it != myvector.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::end**

end - возвращает итератор, указывающий на элемент, следующий за последним элементом контейнера.

#### **vector::rbegin**

rbegin - возвращает обратный итератор, указывающий на последний элемент контейнера.

Пример использования методов rbegin() и rend() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector(5); // 5 целых чисел, построенных по умолчанию

int i = 0;

std::vector<int>::reverse\_iterator rit = myvector.rbegin();

for (; rit != myvector.rend(); ++rit)

\*rit = ++i;

std::cout << "myvector contains:";

for (std::vector<int>::iterator it = myvector.begin(); it != myvector.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::rend**

rend - возвращает обратный итератор, указывающий на элемент, предшествующий первому элементу контейнера.

#### **vector::cbegin**

cbegin - возвращает константный итератор, указывающий на первый элемент контейнера.

Пример использования методов cbegin() и cend() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector = { 10,20,30,40,50 };

std::cout << "myvector contains:";

for (auto it = myvector.cbegin(); it != myvector.cend(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::cend**

cend - возвращает константный итератор, указывающий на элемент, следующий за последним элементом контейнера.

#### **vector::crbegin**

crbegin - возвращает константный обратный итератор, указывающий на последний элемент контейнера.

Пример использования методов cbegin() и crend() для обратной итерации по элементам std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector = { 1,2,3,4,5 };

std::cout << "myvector backwards:";

for (auto rit = myvector.crbegin(); rit != myvector.crend(); ++rit)

std::cout << ' ' << \*rit;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::crend**

crend - возвращает константный обратный итератор, указывающий на элемент, предшествующий первому элементу контейнера.

#### **vector::size**

size - возвращает количество элементов в контейнере.

Пример использования метода size() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myints;

std::cout << "0. size: " << myints.size() << '\n';

for (int i = 0; i < 10; i++) myints.push\_back(i);

std::cout << "1. size: " << myints.size() << '\n';

myints.insert(myints.end(), 10, 100);

std::cout << "2. size: " << myints.size() << '\n';

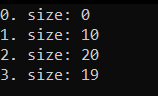
myints.pop\_back();

std::cout << "3. size: " << myints.size() << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::max\_size**

max\_size - возвращает максимально возможное количество элементов, которое может содержать контейнер.

Пример использования метода max\_size() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

for (int i = 0; i < 100; i++) myvector.push\_back(i);

std::cout << "size: " << myvector.size() << "\n";

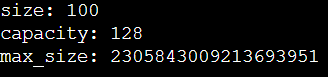
std::cout << "capacity: " << myvector.capacity() << "\n";

std::cout << "max\_size: " << myvector.max\_size() << "\n";

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::empty**

empty - проверяет, является ли контейнер пустым.

Пример использования метода empty() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

int sum(0);

for (int i = 1;i <= 10;i++) myvector.push\_back(i);

while (!myvector.empty()) {

sum += myvector.back();

myvector.pop\_back();

}

std::cout << "total: " << sum << '\n';

return 0;

}

В примере содержимое вектора инициализируется последовательностью чисел (от 1 до 10). Затем он извлекает элементы один за другим, пока не станет пустым, и вычисляет их сумму.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::resize**

resize - изменяет размер контейнера, добавляя или удаляя элементы.

Пример использования метода resize() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

for (int i = 1;i < 10;i++) myvector.push\_back(i);

myvector.resize(5);

myvector.resize(8, 100);

myvector.resize(12);

std::cout << "myvector contains:";

for (int i = 0;i < myvector.size();i++)

std::cout << ' ' << myvector[i];

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::shrink\_to\_fit**

shrink\_to\_fit - уменьшает емкость контейнера до его текущего размера.

Пример использования метода shrink\_to\_fit() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector(100);

std::cout << "1. capacity of myvector: " << myvector.capacity() << '\n';

myvector.resize(10);

std::cout << "2. capacity of myvector: " << myvector.capacity() << '\n';

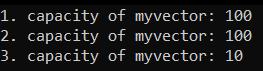
myvector.shrink\_to\_fit();

std::cout << "3. capacity of myvector: " << myvector.capacity() << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::capacity**

capacity - возвращает текущую емкость контейнера.

Пример использования метода capacity() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

for (int i = 0; i < 100; i++) myvector.push\_back(i);

std::cout << "size: " << (int)myvector.size() << '\n';

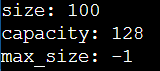
std::cout << "capacity: " << (int)myvector.capacity() << '\n';

std::cout << "max\_size: " << (int)myvector.max\_size() << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::reserve**

reserve - увеличивает емкость контейнера до заданной величины.

Пример использования метода reserve() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int>::size\_type sz;

std::vector<int> foo;

sz = foo.capacity();

std::cout << "making foo grow:\n";

for (int i = 0; i < 100; ++i) {

foo.push\_back(i);

if (sz != foo.capacity()) {

sz = foo.capacity();

std::cout << "capacity changed: " << sz << '\n';

}

}

std::vector<int> bar;

sz = bar.capacity();

bar.reserve(100); // это единственное отличие от приведенного выше foo

std::cout << "making bar grow:\n";

for (int i = 0; i < 100; ++i) {

bar.push\_back(i);

if (sz != bar.capacity()) {

sz = bar.capacity();

std::cout << "capacity changed: " << sz << '\n';

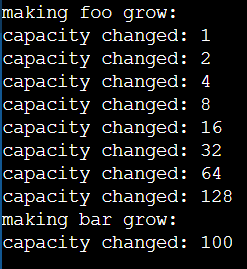
}

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::front**

front - возвращает ссылку на первый элемент контейнера.

Пример использования метода front() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

myvector.push\_back(78);

myvector.push\_back(16);

// теперь спереди равно 78, а сзади 16

myvector.front() -= myvector.back();

std::cout << "myvector.front() is now " << myvector.front() << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::back**

back - возвращает ссылку на последний элемент контейнера.

Пример использования метода back() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

myvector.push\_back(10);

while (myvector.back() != 0) {

myvector.push\_back(myvector.back() - 1);

}

std::cout << "myvector contains:";

for (unsigned i = 0; i < myvector.size(); i++)

std::cout << ' ' << myvector[i];

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::operator[]**

operator[] - обеспечивает доступ к элементам контейнера по индексу.

Пример использования метода operator[] в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector(10); // 10 элементов, инициализированных нулем

std::vector<int>::size\_type sz = myvector.size();

for (unsigned i = 0; i < sz; i++) myvector[i] = i;

// обратный вектор с использованием оператора[]:

for (unsigned i = 0; i < sz / 2; i++) {

int temp;

temp = myvector[sz - 1 - i];

myvector[sz - 1 - i] = myvector[i];

myvector[i] = temp;

}

std::cout << "myvector contains:";

for (unsigned i = 0; i < sz; i++)

std::cout << ' ' << myvector[i];

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::at**

at - обеспечивает доступ к элементам контейнера по индексу с проверкой границ.

Пример использования метода at() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector(10); // 10 инициализированный нулем int

for (unsigned i = 0; i < myvector.size(); i++)

myvector.at(i) = i;

std::cout << "myvector contains:";

for (unsigned i = 0; i < myvector.size(); i++)

std::cout << ' ' << myvector.at(i);

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::assign**

assign - заменяет содержимое контейнера новыми элементами.

Пример использования метода assign() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> first;

std::vector<int> second;

std::vector<int> third;

first.assign(7, 100); // 7 целых чисел со значением 100

std::vector<int>::iterator it;

it = first.begin() + 1;

second.assign(it, first.end() - 1); // 5 основных значений первого

int myints[] = { 1776,7,4 };

third.assign(myints, myints + 3); // присваивание из массива

std::cout << "Size of first: " << int(first.size()) << '\n';

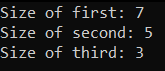
std::cout << "Size of second: " << int(second.size()) << '\n';

std::cout << "Size of third: " << int(third.size()) << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::emplace**

emplace - вставляет элемент с использованием конструктора на месте.

Пример использования метода emplace() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector = { 10,20,30 };

auto it = myvector.emplace(myvector.begin() + 1, 100);

myvector.emplace(it, 200);

myvector.emplace(myvector.end(), 300);

std::cout << "myvector contains:";

for (auto& x : myvector)

std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::insert**

insert - вставляет элементы в контейнер на заданную позицию.

Пример использования метода insert() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector(3, 100);

std::vector<int>::iterator it;

it = myvector.begin();

it = myvector.insert(it, 200);

myvector.insert(it, 2, 300);

// "it" больше недействителен, получите новый:

it = myvector.begin();

std::vector<int> anothervector(2, 400);

myvector.insert(it + 2, anothervector.begin(), anothervector.end());

int myarray[] = { 501,502,503 };

myvector.insert(myvector.begin(), myarray, myarray + 3);

std::cout << "myvector contains:";

for (it = myvector.begin(); it < myvector.end(); it++)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::erase**

erase - удаляет элементы из контейнера по заданной позиции или диапазону.

Пример использования метода erase() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

// установите несколько значений (от 1 до 10)

for (int i = 1; i <= 10; i++) myvector.push\_back(i);

// стереть 6-й элемент

myvector.erase(myvector.begin() + 5);

// стереть первые 3 элемента:

myvector.erase(myvector.begin(), myvector.begin() + 3);

std::cout << "myvector contains:";

for (unsigned i = 0; i < myvector.size(); ++i)

std::cout << ' ' << myvector[i];

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::emplace\_back**

emplace\_back - вставляет элемент в конец контейнера, используя конструктор на месте.

Пример использования метода emplace\_back() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector = { 10,20,30 };

myvector.emplace\_back(100);

myvector.emplace\_back(200);

std::cout << "myvector contains:";

for (auto& x : myvector)

std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::push\_back**

push\_back - добавляет элемент в конец контейнера.

Пример использования метода push\_back() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

int myint;

std::cout << "Please enter some integers (enter 0 to end):\n";

do {

std::cin >> myint;

myvector.push\_back(myint);

} while (myint);

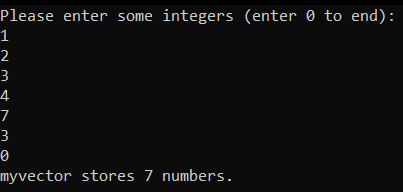
std::cout << "myvector stores " << int(myvector.size()) << " numbers.\n";

return 0;

}

В примере используется push\_back для добавления нового элемента в вектор каждый раз, когда считывается новое целое число.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::pop\_back**

pop\_back - удаляет последний элемент контейнера.

Пример использования метода pop\_back() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

int sum(0);

myvector.push\_back(100);

myvector.push\_back(200);

myvector.push\_back(300);

while (!myvector.empty()) {

sum += myvector.back();

myvector.pop\_back();

}

std::cout << "The elements of myvector add up to " << sum << '\n';

return 0;

}

В этом примере элементы извлекаются из вектора после того, как они суммируются в сумме.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::clear**

clear - удаляет все элементы из контейнера.

Пример использования метода clear() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

myvector.push\_back(100);

myvector.push\_back(200);

myvector.push\_back(300);

std::cout << "myvector contains:";

for (unsigned i = 0; i < myvector.size(); i++)

std::cout << ' ' << myvector[i];

std::cout << '\n';

myvector.clear();

myvector.push\_back(1101);

myvector.push\_back(2202);

std::cout << "myvector contains:";

for (unsigned i = 0; i < myvector.size(); i++)

std::cout << ' ' << myvector[i];

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::swap**

swap - обменивает содержимое двух контейнеров.

Пример использования метода swap() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> foo(3, 100); // три целых числа со значением 100

std::vector<int> bar(5, 200); // пять целых чисел со значением 200

foo.swap(bar);

std::cout << "foo contains:";

for (unsigned i = 0; i < foo.size(); i++)

std::cout << ' ' << foo[i];

std::cout << '\n';

std::cout << "bar contains:";

for (unsigned i = 0; i < bar.size(); i++)

std::cout << ' ' << bar[i];

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::get\_allocator**

get\_allocator - возвращает аллокатор, используемый контейнером.

Пример использования метода get\_allocator() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector;

int\* p;

unsigned int i;

// выделите массив с пространством для 5 элементов, используя распределитель vector:

p = myvector.get\_allocator().allocate(5);

// создавать значения на месте в массиве:

for (i = 0; i < 5; i++) myvector.get\_allocator().construct(&p[i], i);

std::cout << "The allocated array contains:";

for (i = 0; i < 5; i++) std::cout << ' ' << p[i];

std::cout << '\n';

// уничтожить и освободить:

for (i = 0; i < 5; i++) myvector.get\_allocator().destroy(&p[i]);

myvector.get\_allocator().deallocate(p, 5);

return 0;

}

В примере показан сложный способ выделения памяти для массива целых чисел с использованием того же аллокатора, который используется вектором.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **vector::data**

data - возвращает указатель на первый элемент контейнера.

Пример использования метода data() в std::vector:

#include <iostream>

#include <vector>

int main() {

std::vector<int> myvector(5);

int\* p = myvector.data();

\*p = 10;

++p;

\*p = 20;

p[2] = 100;

std::cout << "myvector contains:";

for (unsigned i = 0; i < myvector.size(); ++i)

std::cout << ' ' << myvector[i];

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

## **Deque**

template < class T, class Alloc = allocator<T> > класс deque; // общий шаблон

Дек - это сокращение от "double-ended queue" (очередь с двумя концами). Двусторонние очереди являются контейнерами последовательности с динамическим размером, которые могут быть расширены или сжаты с обоих концов (как спереди, так и сзади).

Конкретные библиотеки могут реализовывать деки по-разному, обычно в виде некоторой формы динамического массива. В любом случае, они позволяют получать прямой доступ к отдельным элементам с помощью итераторов произвольного доступа, а хранение элементов управляется автоматически путем расширения и сокращения контейнера по мере необходимости.

Таким образом, они обеспечивают функциональность, подобную векторам, но с эффективной вставкой и удалением элементов как в начале последовательности, так и в ее конце. Однако, в отличие от векторов, деки не гарантируют хранение всех своих элементов в смежных областях памяти: доступ к элементам в деке путем смещения указателя к другому элементу вызывает неопределенное поведение.

И векторы, и деки предоставляют очень похожий интерфейс и могут использоваться для схожих целей, но внутренне они работают по-разному: в то время как векторы используют один массив, который иногда требует перевыделения памяти для увеличения размера, элементы дека могут располагаться в различных частях памяти, и контейнер внутренне хранит необходимую информацию для прямого доступа к любому из его элементов за константное время и с единым последовательным интерфейсом (через итераторы). Поэтому деки немного более сложны внутренне по сравнению с векторами, но это позволяет им более эффективно расти в определенных ситуациях, особенно с очень длинными последовательностями, где перевыделения памяти становятся более затратными.

Для операций, которые часто включают вставку или удаление элементов в позициях, отличных от начала или конца, деки работают хуже и имеют менее надежные итераторы и ссылки, чем списки и однонаправленные списки.

**Свойства контейнера**:

* *Последовательность*: элементы в контейнерах-последовательностях упорядочены в строгой линейной последовательности. Доступ к отдельным элементам осуществляется по их положению в этой последовательности.
* *Динамический массив*: как правило, реализован как динамический массив, позволяющий прямой доступ к любому элементу в последовательности и обеспечивающий относительно быстрое добавление/удаление элементов в начале или конце последовательности.
* *Ориентированный на распределитель*: контейнер использует объект аллокатора для динамической обработки своих потребностей в памяти.

**Параметры шаблона**:

* T: Тип элементов, содержащихся в деке. Псевдонимом типа является deque::value\_type.
* Alloc: Тип объекта аллокатора, используемого для определения модели выделения памяти. По умолчанию используется шаблон класса аллокатора, который определяет самую простую модель выделения памяти и не зависит от типа значения. Псевдонимом типа является deque::allocator\_type.

#### **deque::begin**

begin - возвращает итератор, указывающий на первый элемент контейнера.

Пример использования метода begin()в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque;

for (int i = 1; i <= 5; i++) mydeque.push\_back(i);

std::cout << "mydeque contains:";

std::deque<int>::iterator it = mydeque.begin();

while (it != mydeque.end())

std::cout << ' ' << \*it++;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::end**

end - возвращает итератор, указывающий на элемент, следующий за последним элементом контейнера.

Пример использования метода end() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque;

for (int i = 1; i <= 5; i++) mydeque.insert(mydeque.end(), i);

std::cout << "mydeque contains:";

std::deque<int>::iterator it = mydeque.begin();

while (it != mydeque.end())

std::cout << ' ' << \*it++;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::rbegin**

rbegin - возвращает обратный итератор, указывающий на последний элемент контейнера.

Пример использования методов rbegin() и rend() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque(5); // 5 целых чисел, построенных по умолчанию

std::deque<int>::reverse\_iterator rit = mydeque.rbegin();

int i = 0;

for (rit = mydeque.rbegin(); rit != mydeque.rend(); ++rit)

\*rit = ++i;

std::cout << "mydeque contains:";

for (std::deque<int>::iterator it = mydeque.begin(); it != mydeque.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::rend**

rend - возвращает обратный итератор, указывающий на элемент, предшествующий первому элементу контейнера.

#### **deque::cbegin**

cbegin - возвращает константный итератор, указывающий на первый элемент контейнера.

Пример использования методов cbegin() и cend() для итерации по элементам std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque = { 10,20,30,40,50 };

std::cout << "mydeque contains:";

for (auto it = mydeque.cbegin(); it != mydeque.cend(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::cend**

cend - возвращает константный итератор, указывающий на элемент, следующий за последним элементом контейнера.

#### **deque::crbegin**

crbegin - возвращает константный обратный итератор, указывающий на последний элемент контейнера.

Пример использования методов cbegin() и crend() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque = { 1,2,3,4,5 };

std::cout << "mydeque backwards:";

for (auto rit = mydeque.crbegin(); rit != mydeque.crend(); ++rit)

std::cout << ' ' << \*rit;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::crend**

crend - возвращает константный обратный итератор, указывающий на элемент, предшествующий первому элементу контейнера.

#### **deque::size**

size - возвращает количество элементов в контейнере.

Пример использования метода size() в std::deque:

#include <iostream>

#include <deque>

int main(){

std::deque<int> myints;

std::cout << "0. size: " << myints.size() << '\n';

for (int i = 0; i < 5; i++) myints.push\_back(i);

std::cout << "1. size: " << myints.size() << '\n';

myints.insert(myints.begin(), 5, 100);

std::cout << "2. size: " << myints.size() << '\n';

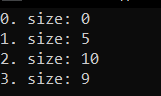
myints.pop\_back();

std::cout << "3. size: " << myints.size() << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::max\_size**

max\_size - возвращает максимально возможное количество элементов, которое может содержать контейнер.

Пример использования метода max\_size() в std::deque:

#include <iostream>

#include <deque>

int main() {

unsigned int i;

std::deque<int> mydeque;

std::cout << "Enter number of elements: ";

std::cin >> i;

if (i < mydeque.max\_size()) mydeque.resize(i);

else std::cout << "That size exceeds the limit.\n";

return 0;

}

Здесь элемент max\_size используется для предварительной проверки того, будет ли запрошенный размер разрешен при изменении.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::empty**

empty - проверяет, является ли контейнер пустым.

Пример использования метода empty() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque;

int sum(0);

for (int i = 1;i <= 10;i++) mydeque.push\_back(i);

while (!mydeque.empty()) {

sum += mydeque.front();

mydeque.pop\_front();

}

std::cout << "total: " << sum << '\n';

return 0;

}

В примере содержимое дека инициализируется последовательностью чисел (от 1 до 10). Затем он извлекает элементы один за другим, пока не станет пустым, и вычисляет их сумму.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::resize**

resize - изменяет размер контейнера, добавляя или удаляя элементы.

Пример использования метода resize() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque;

std::deque<int>::iterator it;

for (int i = 1; i < 10; ++i) mydeque.push\_back(i);

mydeque.resize(5);

mydeque.resize(8, 100);

mydeque.resize(12);

std::cout << "mydeque contains:";

for (std::deque<int>::iterator it = mydeque.begin(); it != mydeque.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Код устанавливает последовательность из 9 чисел в качестве исходного содержимого для mydeque. Затем он сначала использует resize, чтобы установить размер контейнера равным 5, затем увеличить его размер до 8 со значениями 100 для его новых элементов, и, наконец, он увеличивает его размер до 12 со значениями по умолчанию (для элементов int это значение равно нулю).

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::shrink\_to\_fit**

shrink\_to\_fit - уменьшает емкость контейнера до его текущего размера.

Пример использования метода shrink\_to\_fit() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque(100);

std::cout << "1. size of mydeque: " << mydeque.size() << '\n';

mydeque.resize(10);

std::cout << "2. size of mydeque: " << mydeque.size() << '\n';

mydeque.shrink\_to\_fit();

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::front**

front - возвращает ссылку на первый элемент контейнера.

Пример использования метода front() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque;

mydeque.push\_front(77);

mydeque.push\_back(20);

mydeque.front() -= mydeque.back();

std::cout << "mydeque.front() is now " << mydeque.front() << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::back**

back - возвращает ссылку на последний элемент контейнера.

Пример использования метода back() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque;

mydeque.push\_back(10);

while (mydeque.back() != 0)

mydeque.push\_back(mydeque.back() - 1);

std::cout << "mydeque contains:";

for (std::deque<int>::iterator it = mydeque.begin(); it != mydeque.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::operator[]**

operator[] - обеспечивает доступ к элементам контейнера по индексу.

Пример использования метода operator[] в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque(10); // 10 элементов, инициализированных нулем

std::deque<int>::size\_type sz = mydeque.size();

// присвоить некоторые значения:

for (unsigned i = 0; i < sz; i++) mydeque[i] = i;

// обратный порядок элементов с использованием оператора[]:

for (unsigned i = 0; i < sz / 2; i++) {

int temp;

temp = mydeque[sz - 1 - i];

mydeque[sz - 1 - i] = mydeque[i];

mydeque[i] = temp;

}

// вывод:

std::cout << "mydeque contains:";

for (unsigned i = 0; i < sz; i++)

std::cout << ' ' << mydeque[i];

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::at**

at - обеспечивает доступ к элементам контейнера по индексу с проверкой границ.

Пример использования метода at() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<unsigned> mydeque(10); // 10 инициализированных нулем unsigned

// присвоить некоторые значения:

for (unsigned i = 0; i < mydeque.size(); i++)

mydeque.at(i) = i;

std::cout << "mydeque contains:";

for (unsigned i = 0; i < mydeque.size(); i++)

std::cout << ' ' << mydeque.at(i);

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::assign**

assign - заменяет содержимое контейнера новыми элементами.

Пример использования метода assign() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> first;

std::deque<int> second;

std::deque<int> third;

first.assign(7, 100); // 7 целых чисел со значением 100

std::deque<int>::iterator it;

it = first.begin() + 1;

second.assign(it, first.end() - 1); // 5 основных значений первого

int myints[] = { 1776,7,4 };

third.assign(myints, myints + 3); // присваивание из массива

std::cout << "Size of first: " << int(first.size()) << '\n';

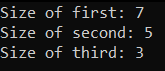
std::cout << "Size of second: " << int(second.size()) << '\n';

std::cout << "Size of third: " << int(third.size()) << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::emplace**

emplace - вставляет элемент с использованием конструктора на месте.

Пример использования метода emplace() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque = { 10,20,30 };

auto it = mydeque.emplace(mydeque.begin() + 1, 100);

mydeque.emplace(it, 200);

mydeque.emplace(mydeque.end(), 300);

std::cout << "mydeque contains:";

for (auto& x : mydeque)

std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::insert**

insert - вставляет элементы в контейнер на заданную позицию.

Пример использования метода insert() в std::deque:

#include <iostream>

#include <deque>

#include <vector>

int main() {

std::deque<int> mydeque;

// установите некоторые начальные значения:

for (int i = 1; i < 6; i++) mydeque.push\_back(i); // 1 2 3 4 5

std::deque<int>::iterator it = mydeque.begin();

++it;

it = mydeque.insert(it, 10); // 1 10 2 3 4 5

// "it" теперь указывает на недавно вставленный 10

mydeque.insert(it, 2, 20); // 1 20 20 10 2 3 4 5

// "это" больше недействительно!

it = mydeque.begin() + 2;

std::vector<int> myvector(2, 30);

mydeque.insert(it, myvector.begin(), myvector.end());

// 1 20 30 30 20 10 2 3 4 5

std::cout << "mydeque contains:";

for (it = mydeque.begin(); it != mydeque.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::erase**

erase - удаляет элементы из контейнера по заданной позиции или диапазону.

Пример использования метода erase() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque;

// установите несколько значений (от 1 до 10)

for (int i = 1; i <= 10; i++) mydeque.push\_back(i);

// стереть 6-й элемент

mydeque.erase(mydeque.begin() + 5);

// сотрите первые 3 элемента:

mydeque.erase(mydeque.begin(), mydeque.begin() + 3);

std::cout << "mydeque contains:";

for (std::deque<int>::iterator it = mydeque.begin(); it != mydeque.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::emplace\_back**

emplace\_back - вставляет элемент в конец контейнера, используя конструктор на месте.

Пример использования метода emplace\_back() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque = { 10,20,30 };

mydeque.emplace\_back(100);

mydeque.emplace\_back(200);

std::cout << "mydeque contains:";

for (auto& x : mydeque)

std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::push\_back**

push\_back - добавляет элемент в конец контейнера.

Пример использования метода push\_back() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque;

int myint;

std::cout << "Please enter some integers (enter 0 to end):\n";

do {

std::cin >> myint;

mydeque.push\_back(myint);

} while (myint);

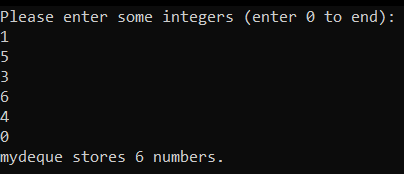
std::cout << "mydeque stores " << (int)mydeque.size() << " numbers.\n";

return 0;

}

В примере используется push\_back для добавления нового элемента в контейнер каждый раз, когда считывается новое целое число.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::pop\_back**

pop\_back - удаляет последний элемент контейнера.

Пример использования метода pop\_back() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque;

int sum(0);

mydeque.push\_back(10);

mydeque.push\_back(20);

mydeque.push\_back(30);

while (!mydeque.empty()) {

sum += mydeque.back();

mydeque.pop\_back();

}

std::cout << "The elements of mydeque add up to " << sum << '\n';

return 0;

}

В этом примере элементы выводятся из конца списка после того, как они суммируются в сумме.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::emplace\_front**

emplace\_front - вставляет элемент в начало контейнера, используя конструктор на месте.

Пример использования метода emplace\_front() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque = { 10,20,30 };

mydeque.emplace\_front(111);

mydeque.emplace\_front(222);

std::cout << "mydeque contains:";

for (auto& x : mydeque)

std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::push\_front**

push\_front - добавляет элемент в начало контейнера.

Пример использования метода push\_front() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque(2, 100); // два целых числа со значением 100

mydeque.push\_front(200);

mydeque.push\_front(300);

std::cout << "mydeque contains:";

for (std::deque<int>::iterator it = mydeque.begin(); it != mydeque.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::pop\_front**

pop\_front - удаляет первый элемент контейнера.

Пример использования метода pop\_front() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque;

mydeque.push\_back(100);

mydeque.push\_back(200);

mydeque.push\_back(300);

std::cout << "Popping out the elements in mydeque:";

while (!mydeque.empty()) {

std::cout << ' ' << mydeque.front();

mydeque.pop\_front();

}

std::cout << "\nThe final size of mydeque is " << int(mydeque.size()) << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::clear**

clear - удаляет все элементы из контейнера.

Пример использования метода clear() в std::deque:

#include <iostream>

#include <deque>

int main() {

unsigned int i;

std::deque<int> mydeque;

mydeque.push\_back(100);

mydeque.push\_back(200);

mydeque.push\_back(300);

std::cout << "mydeque contains:";

for (std::deque<int>::iterator it = mydeque.begin(); it != mydeque.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

mydeque.clear();

mydeque.push\_back(1101);

mydeque.push\_back(2202);

std::cout << "mydeque contains:";

for (std::deque<int>::iterator it = mydeque.begin(); it != mydeque.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::swap**

swap - обменивает содержимое двух контейнеров.

Пример использования метода swap() в std::deque:

#include <iostream>

#include <deque>

main() {

unsigned int i;

std::deque<int> foo(3, 100); // три целых числа со значением 100

std::deque<int> bar(5, 200); // пять целых чисел со значением 200

foo.swap(bar);

std::cout << "foo contains:";

for (std::deque<int>::iterator it = foo.begin(); it != foo.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

std::cout << "bar contains:";

for (std::deque<int>::iterator it = bar.begin(); it != bar.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **deque::get\_allocator**

get\_allocator - возвращает аллокатор, используемый контейнером.

Пример использования метода get\_allocator() в std::deque:

#include <iostream>

#include <deque>

int main() {

std::deque<int> mydeque;

int\* p;

unsigned int i;

// выделите массив с пространством для 5 элементов, используя распределитель deque:

p = mydeque.get\_allocator().allocate(5);

// создавать значения на месте в массиве:

for (i = 0; i < 5; i++) mydeque.get\_allocator().construct(&p[i], i);

std::cout << "The allocated array contains:";

for (i = 0; i < 5; i++) std::cout << ' ' << p[i];

std::cout << '\n';

// уничтожить и освободить:

for (i = 0; i < 5; i++) mydeque.get\_allocator().destroy(&p[i]);

mydeque.get\_allocator().deallocate(p, 5);

return 0;

}

В примере показан сложный способ выделения памяти для массива целых чисел с использованием того же аллокатора, который используется деком.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

## **Forward\_list**

template < class T, class Alloc = allocator<T> > класс forward\_list; // общий шаблон

Forward lists - это контейнеры последовательностей, которые позволяют выполнять операции вставки и удаления в любое время в любом месте последовательности.

Forward lists реализованы в виде односвязных списков; Односвязные списки могут хранить каждый из своих элементов в разных и независимых областях памяти. Порядок элементов сохраняется благодаря связи каждого элемента с ссылкой на следующий элемент в последовательности.

Основное различие между контейнером forward\_list и контейнером list заключается в том, что первый хранит внутри только ссылку на следующий элемент, в то время как последний хранит две ссылки на каждый элемент: одна указывает на следующий элемент, а другая на предыдущий, что обеспечивает эффективную итерацию в обоих направлениях, но требует дополнительной памяти на каждый элемент и немного большего временного затрачивания на вставку и удаление элементов. Таким образом, объекты forward\_list более эффективны, чем объекты list, хотя их можно только итерировать вперед.

По сравнению с другими базовыми стандартными контейнерами последовательностей (array, vector и deque), forward\_list в целом лучше выполняет вставку, извлечение и перемещение элементов в любую позицию контейнера, а, следовательно, и в алгоритмах, которые интенсивно их используют, таких как алгоритмы сортировки.

Основным недостатком forward\_lists и списков по сравнению с этими другими контейнерами последовательностей является отсутствие прямого доступа к элементам по их позиции. Например, чтобы получить доступ к шестому элементу в forward\_list, нужно выполнить итерацию от начала до этой позиции, что занимает линейное время в зависимости от расстояния между ними. Они также требуют дополнительную память для хранения связующей информации, связанной с каждым элементом (что может быть важным фактором для больших списков элементов небольшого размера).

Шаблон класса forward\_list был разработан с учетом эффективности: по замыслу он так же эффективен, как простой односвязный список, написанный вручную на языке C. Фактически, forward\_list является единственным стандартным контейнером, в котором намеренно отсутствует функция-член size в целях повышения эффективности: из-за своей структуры в виде связанного списка, наличие функции-члена size, которая работает за постоянное время, потребовало бы внутреннего счетчика для хранения размера (как это делает list). Это потребовало бы дополнительного использования памяти и сделало бы операции вставки и удаления немного менее эффективными. Чтобы получить размер объекта forward\_list, вы можно использовать алгоритм distance с его begin и end, что является операцией, выполняющейся за линейное время.

**Свойства контейнера**:

* *Последовательность*: элементы в контейнерах-последовательностях упорядочены в строгой линейной последовательности. Доступ к отдельным элементам осуществляется по их положению в этой последовательности.
* *Однонаправленный список*: каждый элемент хранит информацию о том, как найти следующий элемент, что позволяет выполнять операции вставки и удаления за постоянное время после определенного элемента (даже целых диапазонов), но не обеспечивает прямого произвольного доступа..
* *Ориентированный на распределитель*: контейнер использует объект аллокатора для динамической обработки своих потребностей в памяти.

**Параметры шаблона**:

* T: Тип элементов, содержащихся в списке. Псевдонимом типа является forward\_list::value\_type.
* Alloc: Тип объекта аллокатора, используемого для определения модели выделения памяти. По умолчанию используется шаблон класса аллокатора, который определяет самую простую модель выделения памяти и не зависит от типа значения. Псевдонимом типа является forward\_list::allocator\_type.

#### **forward\_list::begin**

begin - возвращает итератор, указывающий на первый элемент контейнера.

Пример использования метода begin() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 34, 77, 16, 2 };

std::cout << "mylist contains:";

for (auto it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::before\_begin**

Пример использования метода before\_begin() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 20, 30, 40, 50 };

mylist.insert\_after(mylist.before\_begin(), 11);

std::cout << "mylist contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::end**

end - возвращает итератор, указывающий на элемент, следующий за последним элементом контейнера.

Пример использования метода end() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 10, 20, 30, 40 };

std::cout << "mylist contains:";

for (auto it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::cbegin**

cbegin - возвращает константный итератор, указывающий на первый элемент контейнера.

Пример использования методов cbegin() и cend() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 21, 32, 12 };

std::cout << "myarray contains:";

for (auto it = mylist.cbegin(); it != mylist.cend(); ++it)

std::cout << ' ' << \*it; // не удается изменить \*it

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::cbefore\_begin**

Пример использования метода cbefore\_begin() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 77, 2, 16 };

mylist.insert\_after(mylist.cbefore\_begin(), 19);

std::cout << "mylist contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::cend**

cend - возвращает константный итератор, указывающий на элемент, следующий за последним элементом контейнера.

#### **forward\_list::max\_size**

max\_size - возвращает максимально возможное количество элементов, которое может содержать контейнер.

Пример использования метода max\_size() в std::forward\_list:

#include <iostream>

#include <sstream>

#include <forward\_list>

int main() {

int myint;

std::string mystring;

std::forward\_list<int> mylist;

std::cout << "Enter size: ";

std::getline(std::cin, mystring);

std::stringstream(mystring) >> myint;

if (myint <= mylist.max\_size()) mylist.resize(myint);

else std::cout << "That size exceeds the maximum.\n";

return 0;

}

Здесь элемент max\_size используется для предварительной проверки того, будет ли запрошенный размер разрешен при изменении размера элемента.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::empty**

empty - проверяет, является ли контейнер пустым.

Пример использования метода empty() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> first;

std::forward\_list<int> second = { 20, 40, 80 };

std::cout << "first " << (first.empty() ? "is empty" : "is not empty") << std::endl;

std::cout << "second " << (second.empty() ? "is empty" : "is not empty") << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::resize**

resize - изменяет размер контейнера, добавляя или удаляя элементы.

Пример использования метода resize() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 10, 20, 30, 40, 50 };

// 10 20 30 40 50

mylist.resize(3); // 10 20 30

mylist.resize(5, 100); // 10 20 30 100 100

std::cout << "mylist contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::front**

front - возвращает ссылку на первый элемент контейнера.

Пример использования метода front() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 2, 16, 77 };

mylist.front() = 11;

std::cout << "mylist now contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::assign**

assign - заменяет содержимое контейнера новыми элементами.

Пример использования метода assign() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> first;

std::forward\_list<int> second;

first.assign(4, 15); // 15 15 15 15

second.assign(first.begin(), first.end()); // 15 15 15 15

first.assign({ 77, 2, 16 }); // 77 2 16

std::cout << "first contains: ";

for (int& x : first) std::cout << ' ' << x;

std::cout << '\n';

std::cout << "second contains: ";

for (int& x : second) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::emplace\_after**

Пример использования метода emplace\_after() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list< std::pair<int, char> > mylist;

auto it = mylist.before\_begin();

it = mylist.emplace\_after(it, 100, 'x');

it = mylist.emplace\_after(it, 200, 'y');

it = mylist.emplace\_after(it, 300, 'z');

std::cout << "mylist contains:";

for (auto& x : mylist)

std::cout << " (" << x.first << "," << x.second << ")";

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::insert\_after**

Пример использования метода insert\_after() в std::forward\_list:

#include <iostream>

#include <array>

#include <forward\_list>

int main() {

std::array<int, 3> myarray = { 11, 22, 33 };

std::forward\_list<int> mylist;

std::forward\_list<int>::iterator it;

it = mylist.insert\_after(mylist.before\_begin(), 10); // 10

// ^ <- it

it = mylist.insert\_after(it, 2, 20); // 10 20 20

// ^

it = mylist.insert\_after(it, myarray.begin(), myarray.end());// 10 20 20 11 22 33

// ^

it = mylist.begin(); // ^

it = mylist.insert\_after(it, { 1,2,3 }); // 10 1 2 3 20 20 11 22 33

// ^

std::cout << "mylist contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::erase\_after**

Пример использования метода erase\_after() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 10, 20, 30, 40, 50 };

// 10 20 30 40 50

auto it = mylist.begin(); // ^

it = mylist.erase\_after(it); // 10 30 40 50

// ^

it = mylist.erase\_after(it, mylist.end()); // 10 30

// ^

std::cout << "mylist contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::emplace\_front**

emplace\_front - вставляет элемент в начало контейнера, используя конструктор на месте.

Пример использования метода emplace\_front() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list< std::pair<int, char> > mylist;

mylist.emplace\_front(10, 'a');

mylist.emplace\_front(20, 'b');

mylist.emplace\_front(30, 'c');

std::cout << "mylist contains:";

for (auto& x : mylist)

std::cout << " (" << x.first << "," << x.second << ")";

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::push\_front**

push\_front - добавляет элемент в начало контейнера.

Пример использования метода push\_front() в std::forward\_list:

#include <iostream>

#include <forward\_list>

using namespace std;

int main() {

forward\_list<int> mylist = { 77, 2, 16 };

mylist.push\_front(19);

mylist.push\_front(34);

std::cout << "mylist contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::pop\_front**

pop\_front - удаляет первый элемент контейнера.

Пример использования метода pop\_front() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 10, 20, 30, 40 };

std::cout << "Popping out the elements in mylist:";

while (!mylist.empty()) {

std::cout << ' ' << mylist.front();

mylist.pop\_front();

}

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::clear**

clear - удаляет все элементы из контейнера.

Пример использования метода clear() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 10, 20, 30 };

std::cout << "mylist contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

mylist.clear();

mylist.insert\_after(mylist.before\_begin(), { 100, 200 });

std::cout << "mylist contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::swap**

swap - обменивает содержимое двух контейнеров.

Пример использования метода swap() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> first = { 10, 20, 30 };

std::forward\_list<int> second = { 100, 200 };

std::forward\_list<int>::iterator it;

first.swap(second);

std::cout << "first contains:";

for (int& x : first) std::cout << ' ' << x;

std::cout << '\n';

std::cout << "second contains:";

for (int& x : second) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::splice\_after**

Пример использования метода splice\_after() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> first = { 1, 2, 3 };

std::forward\_list<int> second = { 10, 20, 30 };

auto it = first.begin(); // указывает на 1

first.splice\_after(first.before\_begin(), second);

// первый: 10 20 30 1 2 3

// второй: (пусто)

// "it" по-прежнему указывает на 1 (теперь это 4-й элемент first)

second.splice\_after(second.before\_begin(), first, first.begin(), it);

// первый: 10 1 2 3

// второй: 20 30

first.splice\_after(first.before\_begin(), second, second.begin());

// первый: 30 10 1 2 3

// второй: 20

// \* обратите внимание, что то, что перемещается, находится после итератора

std::cout << "first contains:";

for (int& x : first) std::cout << " " << x;

std::cout << std::endl;

std::cout << "second contains:";

for (int& x : second) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::remove**

remove - удаляет все элементы, равные заданному значению.

Пример использования метода remove() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 10, 20, 30, 40, 30, 20, 10 };

mylist.remove(20);

std::cout << "mylist contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::remove\_if**

remove\_if - удаляет элементы, удовлетворяющие заданному предикату.

Пример использования метода remove\_if() в std::forward\_list:

#include <iostream>

#include <forward\_list>

// предикат, реализованный как функция:

bool single\_digit(const int& value) { return (value < 10); }

// предикат, реализованный в виде класса:

class is\_odd\_class {

public:

bool operator() (const int& value) { return (value % 2) == 1; }

} is\_odd\_object;

int main() {

std::forward\_list<int> mylist = { 7, 80, 7, 15, 85, 52, 6 };

mylist.remove\_if(single\_digit); // 80 15 85 52

mylist.remove\_if(is\_odd\_object); // 80 52

std::cout << "mylist contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::unique**

unique - удаляет все дублирующиеся элементы из контейнера.

Пример использования метода unique() в std::forward\_list:

#include <iostream>

#include <cmath>

#include <forward\_list>

// двоичный предикат, реализованный как функция:

bool same\_integral\_part(double first, double second) {

return (int(first) == int(second));

}

// двоичный предикат, реализованный в виде класса:

class is\_near\_class {

public:

bool operator() (double first, double second) {

return (fabs(first - second) < 5.0);

}

} is\_near\_object;

int main() {

std::forward\_list<double> mylist = { 15.2, 73.0, 3.14, 15.85, 69.5,

73.0, 3.99, 15.2, 69.2, 18.5 };

mylist.sort(); // 3.14, 3.99, 15.2, 15.2, 15.85

// 18.5, 69.2, 69.5, 73.0, 73.0

mylist.unique(); // 3.14, 3.99, 15.2, 15.85

// 18.5, 69.2, 69.5, 73.0

mylist.unique(same\_integral\_part); // 3.14, 15.2, 18.5, 69.2, 73.0

mylist.unique(is\_near\_object); // 3.14, 15.2, 69.2

std::cout << "mylist contains:";

for (double& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::merge**

merge - объединяет два отсортированных контейнера в один отсортированный.

Пример использования метода merge() в std::forward\_list:

#include <iostream>

#include <forward\_list>

#include <functional>

int main() {

std::forward\_list<double> first = { 4.2, 2.9, 3.1 };

std::forward\_list<double> second = { 1.4, 7.7, 3.1 };

std::forward\_list<double> third = { 6.2, 3.7, 7.1 };

first.sort();

second.sort();

first.merge(second);

std::cout << "first contains:";

for (double& x : first) std::cout << " " << x;

std::cout << std::endl;

first.sort(std::greater<double>());

third.sort(std::greater<double>());

first.merge(third, std::greater<double>());

std::cout << "first contains:";

for (double& x : first) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::sort**

sort - сортирует элементы контейнера по возрастанию.

Пример использования метода sort() в std::forward\_list:

#include <iostream>

#include <forward\_list>

#include <functional>

int main() {

std::forward\_list<int> mylist = { 22, 13, 5, 40, 90, 62, 31 };

mylist.sort();

std::cout << "default sort (operator<):";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

mylist.sort(std::greater<int>());

std::cout << "sort with std::greater():";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::reverse**

Пример использования метода reverse() в std::forward\_list:

#include <iostream>

#include <forward\_list>

int main() {

std::forward\_list<int> mylist = { 10, 20, 30, 40 };

mylist.reverse();

std::cout << "mylist contains:";

for (int& x : mylist) std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **forward\_list::get\_allocator**

get\_allocator - возвращает аллокатор, используемый контейнером.

## **List**

template < class T, class Alloc = allocator<T> >класс list; // общий шаблон

Lists - это контейнеры последовательностей, которые позволяют выполнять операции вставки и удаления в любое время в любом месте последовательности, а также итерацию в обоих направлениях.

Контейнер list реализован в виде двусвязного списка. Двусвязные списки могут хранить каждый из своих элементов в разных и независимых областях памяти. Порядок элементов поддерживается внутри контейнера благодаря связи каждого элемента с предыдущим и следующим элементами.

Они очень похожи на контейнер forward\_list: основное отличие состоит в том, что объекты forward\_list являются односвязными списками и могут только итерироваться вперед, взамен они более компактные и эффективные.

По сравнению с другими базовыми стандартными контейнерами последовательностей (array, vector и deque) списки, как правило, лучше выполняют вставку, извлечение и перемещение элементов в любое положение внутри контейнера, для которого уже получен итератор, и, следовательно, также в алгоритмах, которые интенсивно используют их, таких как алгоритмы сортировки.

Основным недостатком списков и forward\_lists по сравнению с этими другими контейнерами последовательностей является отсутствие прямого доступа к элементам по их позиции. Например, чтобы получить доступ к шестому элементу в списке, нужно выполнить итерацию от известной позиции (например, начала или конца) до этой позиции, которая занимает линейное время в зависимости от расстоянии между ними. Они также требуют дополнительную память для хранения связующей информации, связанной с каждым элементом (что может быть важным фактором для больших списков элементов небольшого размера).

**Свойства контейнера**:

* *Последовательность*: элементы в контейнерах-последовательностях упорядочены в строгой линейной последовательности. Доступ к отдельным элементам осуществляется по их положению в этой последовательности.
* *Двухсвязный список*: каждый элемент хранит информацию о том, как найти следующий и предыдущий элемент, что позволяет выполнять операции вставки и удаления за постоянное время после определенного элемента (даже целых диапазонов), но не обеспечивает прямого произвольного доступа.
* *Ориентированный на распределитель*: контейнер использует объект аллокатора для динамической обработки своих потребностей в памяти.

**Параметры шаблона**:

* T: Тип элементов, содержащихся в списке. Псевдонимом типа является list::value\_type.
* Alloc: Тип объекта аллокатора, используемого для определения модели выделения памяти. По умолчанию используется шаблон класса аллокатора, который определяет самую простую модель выделения памяти и не зависит от типа значения. Псевдонимом типа является list::allocator\_type.

#### **list::begin**

Пример использования метода begin() и end() в std::list:

#include <iostream>

#include <list>

int main() {

int myints[] = { 75,23,65,42,13 };

std::list<int> mylist(myints, myints + 5);

std::cout << "mylist contains:";

for (std::list<int>::iterator it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::end**

#### **list::rbegin**

Пример использования методов rbegin() и rend в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

for (int i = 1; i <= 5; ++i) mylist.push\_back(i);

std::cout << "mylist backwards:";

for (std::list<int>::reverse\_iterator rit = mylist.rbegin(); rit != mylist.rend(); ++rit)

std::cout << ' ' << \*rit;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::rend**

#### **list::cbegin**

Пример использования методов cbegin() и cend() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist = { 5,10,15,20 };

std::cout << "mylist contains:";

for (auto it = mylist.cbegin(); it != mylist.cend(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::cend**

#### **list::crbegin**

Пример использования методов cbegin() и crend() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist = { 1,2,4,8,16 };

std::cout << "mylist backwards:";

for (auto rit = mylist.crbegin(); rit != mylist.crend(); ++rit)

std::cout << ' ' << \*rit;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::crend**

#### **list::size**

Пример использования метода size() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> myints;

std::cout << "0. size: " << myints.size() << '\n';

for (int i = 0; i < 10; i++) myints.push\_back(i);

std::cout << "1. size: " << myints.size() << '\n';

myints.insert(myints.begin(), 10, 100);

std::cout << "2. size: " << myints.size() << '\n';

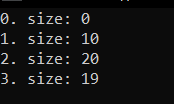
myints.pop\_back();

std::cout << "3. size: " << myints.size() << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::max\_size**

Пример использования метода max\_size() в std::list:

#include <iostream>

#include <list>

int main() {

unsigned int i;

std::list<int> mylist;

std::cout << "Enter number of elements: ";

std::cin >> i;

if (i < mylist.max\_size()) mylist.resize(i);

else std::cout << "That size exceeds the limit.\n";

return 0;

}

Здесь элемент max\_size используется для предварительной проверки того, будет ли запрошенный размер разрешен при изменении элемента.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::empty**

Пример использования метода empty() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

int sum(0);

for (int i = 1;i <= 10;++i) mylist.push\_back(i);

while (!mylist.empty()) {

sum += mylist.front();

mylist.pop\_front();

}

std::cout << "total: " << sum << '\n';

return 0;

}

В примере содержимое контейнера инициализируется последовательностью чисел (от 1 до 10). Затем он извлекает элементы один за другим, пока не станет пустым, и вычисляет их сумму.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::resize**

Пример использования метода resize() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

// установите некоторое начальное содержимое:

for (int i = 1; i < 10; ++i) mylist.push\_back(i);

mylist.resize(5);

mylist.resize(8, 100);

mylist.resize(12);

std::cout << "mylist contains:";

for (std::list<int>::iterator it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Код устанавливает последовательность из 9 чисел в качестве начального содержимого для моего списка. Затем он сначала использует resize, чтобы установить размер контейнера равным 5, затем увеличить его размер до 8 со значениями 100 для его новых элементов, и, наконец, он увеличивает его размер до 12 со значениями по умолчанию (для элементов int это значение равно нулю).

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::front**

Пример использования метода front() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

mylist.push\_back(77);

mylist.push\_back(22);

// теперь спереди равно 77, а сзади 22

mylist.front() -= mylist.back();

std::cout << "mylist.front() is now " << mylist.front() << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::back**

Пример использования метода back() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

mylist.push\_back(10);

while (mylist.back() != 0) {

mylist.push\_back(mylist.back() - 1);

}

std::cout << "mylist contains:";

for (std::list<int>::iterator it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::assign**

Пример использования метода assign() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> first;

std::list<int> second;

first.assign(7, 100); // 7 целых чисел со значением 100

second.assign(first.begin(), first.end()); // копия первого

int myints[] = { 1776,7,4 };

first.assign(myints, myints + 3); // присваивание из массива

std::cout << "Size of first: " << int(first.size()) << '\n';

std::cout << "Size of second: " << int(second.size()) << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::emplace**

Пример использования метода emplace() в std::list:

#include <iostream>

#include <list>

int main() {

std::list< std::pair<int, char> > mylist;

mylist.emplace(mylist.begin(), 100, 'x');

mylist.emplace(mylist.begin(), 200, 'y');

std::cout << "mylist contains:";

for (auto& x : mylist)

std::cout << " (" << x.first << "," << x.second << ")";

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::insert**

Пример использования метода insert() в std::list:

#include <iostream>

#include <list>

#include <vector>

int main() {

std::list<int> mylist;

std::list<int>::iterator it;

// установите некоторые начальные значения:

for (int i = 1; i <= 5; ++i) mylist.push\_back(i); // 1 2 3 4 5

it = mylist.begin();

++it; // теперь он указывает на цифру 2 ^

mylist.insert(it, 10); // 1 10 2 3 4 5

// "it" по-прежнему указывает на номер 2 ^

mylist.insert(it, 2, 20); // 1 10 20 20 2 3 4 5

--it; // теперь это указывает на вторую 20 ^

std::vector<int> myvector(2, 30);

mylist.insert(it, myvector.begin(), myvector.end());

// 1 10 20 30 30 20 2 3 4 5

// ^

std::cout << "mylist contains:";

for (it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::erase**

Пример использования метода erase() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

std::list<int>::iterator it1, it2;

// установите некоторые значения:

for (int i = 1; i < 10; ++i) mylist.push\_back(i \* 10);

// 10 20 30 40 50 60 70 80 90

it1 = it2 = mylist.begin();// ^^

advance(it2, 6); // ^ ^

++it1; // ^ ^

it1 = mylist.erase(it1); // 10 30 40 50 60 70 80 90

// ^ ^

it2 = mylist.erase(it2); // 10 30 40 50 60 80 90

// ^ ^

++it1; // ^ ^

--it2; // ^ ^

mylist.erase(it1, it2); // 10 30 60 80 90

// ^

std::cout << "mylist contains:";

for (it1 = mylist.begin(); it1 != mylist.end(); ++it1)

std::cout << ' ' << \*it1;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::emplace\_back**

Пример использования метода emplace\_back() в std::list:

#include <iostream>

#include <list>

int main() {

std::list< std::pair<int, char> > mylist;

mylist.emplace\_back(10, 'a');

mylist.emplace\_back(20, 'b');

mylist.emplace\_back(30, 'c');

std::cout << "mylist contains:";

for (auto& x : mylist)

std::cout << " (" << x.first << "," << x.second << ")";

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::push\_back**

Пример использования метода push\_back() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

int myint;

std::cout << "Please enter some integers (enter 0 to end):\n";

do {

std::cin >> myint;

mylist.push\_back(myint);

} while (myint);

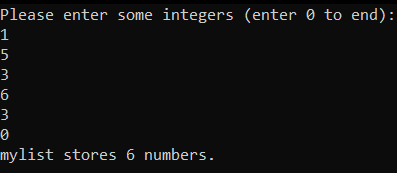
std::cout << "mylist stores " << mylist.size() << " numbers.\n";

return 0;

}

В примере используется push\_back для добавления нового элемента в контейнер каждый раз, когда считывается новое целое число.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::pop\_back**

Пример использования метода pop\_back() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

int sum(0);

mylist.push\_back(100);

mylist.push\_back(200);

mylist.push\_back(300);

while (!mylist.empty()) {

sum += mylist.back();

mylist.pop\_back();

}

std::cout << "The elements of mylist summed " << sum << '\n';

return 0;

}

В этом примере элементы выводятся из конца списка после того, как они суммируются в сумме.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::emplace\_front**

Пример использования метода emplace\_front() в std::list:

#include <iostream>

#include <list>

int main() {

std::list< std::pair<int, char> > mylist;

mylist.emplace\_front(10, 'a');

mylist.emplace\_front(20, 'b');

mylist.emplace\_front(30, 'c');

std::cout << "mylist contains:";

for (auto& x : mylist)

std::cout << " (" << x.first << "," << x.second << ")";

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::push\_front**

Пример использования метода push\_front() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist(2, 100); // два целых числа со значением 100

mylist.push\_front(200);

mylist.push\_front(300);

std::cout << "mylist contains:";

for (std::list<int>::iterator it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::pop\_front**

Пример использования метода pop\_front() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

mylist.push\_back(100);

mylist.push\_back(200);

mylist.push\_back(300);

std::cout << "Popping out the elements in mylist:";

while (!mylist.empty()) {

std::cout << ' ' << mylist.front();

mylist.pop\_front();

}

std::cout << "\nFinal size of mylist is " << mylist.size() << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::clear**

Пример использования метода clear() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

std::list<int>::iterator it;

mylist.push\_back(100);

mylist.push\_back(200);

mylist.push\_back(300);

std::cout << "mylist contains:";

for (it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

mylist.clear();

mylist.push\_back(1101);

mylist.push\_back(2202);

std::cout << "mylist contains:";

for (it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::swap**

Пример использования метода swap() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> first(3, 100); // три целых числа со значением 100

std::list<int> second(5, 200); // пять целых чисел со значением 200

first.swap(second);

std::cout << "first contains:";

for (std::list<int>::iterator it = first.begin(); it != first.end(); it++)

std::cout << ' ' << \*it;

std::cout << '\n';

std::cout << "second contains:";

for (std::list<int>::iterator it = second.begin(); it != second.end(); it++)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::splice**

Пример использования метода splice() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist1, mylist2;

std::list<int>::iterator it;

// установите некоторые начальные значения:

for (int i = 1; i <= 4; ++i)

mylist1.push\_back(i); // mylist1: 1 2 3 4

for (int i = 1; i <= 3; ++i)

mylist2.push\_back(i \* 10); // mylist2: 10 20 30

it = mylist1.begin();

++it; // указывает на 2

mylist1.splice(it, mylist2); // mylist1: 1 10 20 30 2 3 4

// mylist2 (empty)

// "it" по-прежнему указывает на 2 (5-й элемент)

mylist2.splice(mylist2.begin(), mylist1, it);

// mylist1: 1 10 20 30 3 4

// mylist2: 2

// "it" теперь недействительно

it = mylist1.begin();

std::advance(it, 3); // "it" теперь указывает на 30

mylist1.splice(mylist1.begin(), mylist1, it, mylist1.end());

// mylist1: 30 3 4 1 10 20

std::cout << "mylist1 contains:";

for (it = mylist1.begin(); it != mylist1.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

std::cout << "mylist2 contains:";

for (it = mylist2.begin(); it != mylist2.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::remove**

Пример использования метода remove() в std::list:

#include <iostream>

#include <list>

int main() {

int myints[] = { 17,89,7,14 };

std::list<int> mylist(myints, myints + 4);

mylist.remove(89);

std::cout << "mylist contains:";

for (std::list<int>::iterator it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::remove\_if**

Пример использования метода remove\_if() в std::list:

#include <iostream>

#include <list>

// предикат, реализованный как функция:

bool single\_digit(const int& value) { return (value < 10); }

// предикат, реализованный в виде класса:

struct is\_odd {

bool operator() (const int& value) { return (value % 2) == 1; }

};

int main() {

int myints[] = { 15,36,7,17,20,39,4,1 };

std::list<int> mylist(myints, myints + 8); // 15 36 7 17 20 39 4 1

mylist.remove\_if(single\_digit); // 15 36 17 20 39

mylist.remove\_if(is\_odd()); // 36 20

std::cout << "mylist contains:";

for (std::list<int>::iterator it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::unique**

Пример использования метода unique() в std::list:

#include <iostream>

#include <cmath>

#include <list>

// двоичный предикат, реализованный как функция:

bool same\_integral\_part(double first, double second) {

return (int(first) == int(second));

}

// двоичный предикат, реализованный в виде класса:

struct is\_near {

bool operator() (double first, double second) {

return (fabs(first - second) < 5.0);

}

};

int main() {

double mydoubles[] = { 12.15, 2.72, 73.0, 12.77, 3.14,

12.77, 73.35, 72.25, 15.3, 72.25 };

std::list<double> mylist(mydoubles, mydoubles + 10);

mylist.sort(); // 2.72, 3.14, 12.15, 12.77, 12.77,

// 15.3, 72.25, 72.25, 73.0, 73.35

mylist.unique(); // 2.72, 3.14, 12.15, 12.77

// 15.3, 72.25, 73.0, 73.35

mylist.unique(same\_integral\_part); // 2.72, 3.14, 12.15

// 15.3, 72.25, 73.0

mylist.unique(is\_near()); // 2.72, 12.15, 72.25

std::cout << "mylist contains:";

for (std::list<double>::iterator it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::merge**

Пример использования метода merge() в std::list:

#include <iostream>

#include <list>

// сравнивать только неотъемлемую часть:

bool mycomparison(double first, double second) {

return (int(first) < int(second));

}

int main() {

std::list<double> first, second;

first.push\_back(3.1);

first.push\_back(2.2);

first.push\_back(2.9);

second.push\_back(3.7);

second.push\_back(7.1);

second.push\_back(1.4);

first.sort();

second.sort();

first.merge(second);

// (второй теперь пуст)

second.push\_back(2.1);

first.merge(second, mycomparison);

std::cout << "first contains:";

for (std::list<double>::iterator it = first.begin(); it != first.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::sort**

Пример использования метода sort() в std::list:

#include <iostream>

#include <list>

#include <string>

#include <cctype>

// сравнение, не учитывающее регистр

bool compare\_nocase(const std::string& first, const std::string& second) {

unsigned int i = 0;

while ((i < first.length()) && (i < second.length())) {

if (tolower(first[i]) < tolower(second[i])) return true;

else if (tolower(first[i]) > tolower(second[i])) return false;

++i;

}

return (first.length() < second.length());

}

int main() {

std::list<std::string> mylist;

std::list<std::string>::iterator it;

mylist.push\_back("one");

mylist.push\_back("two");

mylist.push\_back("Three");

mylist.sort();

std::cout << "mylist contains:";

for (it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

mylist.sort(compare\_nocase);

std::cout << "mylist contains:";

for (it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Для строк по умолчанию сравнение представляет собой строгое сравнение кодов символов, где все заглавные буквы сравниваются ниже, чем все строчные буквы, при этом все строки, начинающиеся с заглавной буквы, помещаются перед первой операцией сортировки.

При использовании функции compare\_nocase сравнение производится без учета регистра.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::reverse**

Пример использования метода reverse() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

for (int i = 1; i < 10; ++i) mylist.push\_back(i);

mylist.reverse();

std::cout << "mylist contains:";

for (std::list<int>::iterator it = mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

#### **list::ger\_allocator**

Пример использования метода get\_allocator() в std::list:

#include <iostream>

#include <list>

int main() {

std::list<int> mylist;

int\* p;

// выделите массив из 5 элементов, используя распределитель mylist:

p = mylist.get\_allocator().allocate(5);

// присвоить массиву некоторые значения

for (int i = 0; i < 5; ++i) p[i] = i;

std::cout << "The allocated array contains:";

for (int i = 0; i < 5; ++i) std::cout << ' ' << p[i];

std::cout << '\n';

mylist.get\_allocator().deallocate(p, 5);

return 0;

}

В примере показан сложный способ выделения памяти для массива целых чисел с использованием того же распределителя, который используется контейнером.

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры)

# **Ассоциативные контейнеры**

## **Set**

### **Set**

template < class T, // set::key\_type/value\_type class Compare = less<T>, // set::key\_compare/value\_compare class Alloc = allocator<T> // set::allocator\_type > класс set;

Шаблон класса set представляет собой контейнер, который хранит уникальные элементы в определенном порядке.

В set значение элемента также служит его идентификатором (ключом) типа T, и каждое значение должно быть уникальным. Значение элементов в set не может быть изменено после добавления в контейнер (элементы всегда являются константами), но их можно добавлять или удалять из контейнера.

Внутренне элементы в set всегда отсортированы в соответствии с определенным строгим условием слабого упорядочения, указанным в его внутреннем объекте сравнения (тип Compare).

Контейнеры set обычно медленнее контейнеров unordered\_set при доступе к отдельным элементам по их ключу, но позволяют прямую итерацию по подмножествам на основе их порядка.

set обычно реализуется в виде бинарного дерева поиска.

**Свойства контейнера**:

* *Ассоциативность*: элементы в ассоциативных контейнерах ссылается на свои ключи, а не на абсолютную позицию в контейнере.
* *Упорядоченность*: элементы в контейнере всегда следуют строгому порядку. Все добавленные элементы получают позицию в этом порядке.
* *Множество*: значение элемента также используется в качестве ключа для его идентификации.
* *Уникальные ключи*: в контейнере не может быть двух элементов с эквивалентными ключами.
* *Поддержка аллокатора*: контейнер использует объект аллокатора для динамического управления своими потребностями в памяти.

**Параметры шаблона**:

* T: Тип элементов. Каждый элемент в контейнере set также уникально идентифицируется этим значением (каждое значение также является ключом элемента). Алиасом являются член-типы set::key\_type и set::value\_type.
* Compare: Бинарный предикат, который принимает два аргумента того же типа, что и элементы, и возвращает значение типа bool. Выражение comp(a,b), где comp - объект этого типа, а a и b - ключевые значения, должно возвращать true, если a считается предшествующим b в строгом упорядочении, определенном этойфункцией сравнения. Объект типа set использует это выражение для определения порядка следования элементов в контейнере и для проверки эквивалентности двух ключей элементов (сравнивая их рефлексивно: они эквивалентны, если !comp(a,b) && !comp(b,a)). В контейнере set не могут быть два эквивалентных элемента.

Это может быть указатель на функцию или объект функции (см. пример в конструкторе). По умолчанию используется less<T>, которая возвращает результат применения оператора <.

Алиасом являются член-типы set::key\_compare и set::value\_compare.

* Alloc: Тип объекта аллокатора, используемого для определения модели выделения памяти. По умолчанию используется шаблон класса аллокатора, который определяет самую простую модель выделения памяти и не зависит от значения. Алиасом является член-тип set::allocator\_type.

#### **set::begin**

Пример использования методов begin() и end() в std::set:

#include <iostream>

#include <set>

int main() {

int myints[] = { 75,23,65,42,13 };

std::set<int> myset(myints, myints + 5);

std::cout << "myset contains:";

for (std::set<int>::iterator it = myset.begin(); it != myset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::end**

#### **set::rbegin**

Пример использования методов rbegin() и rend() в std::set:

#include <iostream>

#include <set>

int main() {

int myints[] = { 21,64,17,78,49 };

std::set<int> myset(myints, myints + 5);

std::set<int>::reverse\_iterator rit;

std::cout << "myset contains:";

for (rit = myset.rbegin(); rit != myset.rend(); ++rit)

std::cout << ' ' << \*rit;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::rend**

#### **set::cbegin**

Пример использования методов cbegin() и cend() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset = { 50,20,60,10,25 };

std::cout << "myset contains:";

for (auto it = myset.cbegin(); it != myset.cend(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::cend**

#### **set::crbegin**

Пример использования методов crbegin() и crend() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset = { 50,20,60,10,25 };

std::cout << "myset backwards:";

for (auto rit = myset.crbegin(); rit != myset.crend(); ++rit)

std::cout << ' ' << \*rit;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::crend**

#### **set::size**

Пример использования метода size() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myints;

std::cout << "0. size: " << myints.size() << '\n';

for (int i = 0; i < 10; ++i) myints.insert(i);

std::cout << "1. size: " << myints.size() << '\n';

myints.insert(100);

std::cout << "2. size: " << myints.size() << '\n';

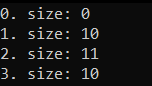
myints.erase(5);

std::cout << "3. size: " << myints.size() << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::max\_size**

Пример использования метода max\_size() в std::set:

#include <iostream>

#include <set>

int main() {

int i;

std::set<int> myset;

if (myset.max\_size() > 1000) {

for (i = 0; i < 1000; i++) myset.insert(i);

std::cout << "The set contains 1000 elements.\n";

}

else std::cout << "The set could not hold 1000 elements.\n";

return 0;

}

Здесь элемент max\_size используется для предварительной проверки того, позволит ли набор вставить 1000 элементов.

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::empty**

Пример использования метода empty() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset;

myset.insert(20);

myset.insert(30);

myset.insert(10);

std::cout << "myset contains:";

while (!myset.empty()) {

std::cout << ' ' << \*myset.begin();

myset.erase(myset.begin());

}

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::emplace**

Пример использования метода emplace() в std::set:

#include <iostream>

#include <set>

#include <string>

int main() {

std::set<std::string> myset;

myset.emplace("foo");

myset.emplace("bar");

auto ret = myset.emplace("foo");

if (!ret.second) std::cout << "foo already exists in myset\n";

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::emplace\_hint**

Пример использования метода emplace\_hint() в std::set:

#include <iostream>

#include <set>

#include <string>

int main() {

std::set<std::string> myset;

auto it = myset.cbegin();

myset.emplace\_hint(it, "alpha");

it = myset.emplace\_hint(myset.cend(), "omega");

it = myset.emplace\_hint(it, "epsilon");

it = myset.emplace\_hint(it, "beta");

std::cout << "myset contains:";

for (const std::string& x : myset)

std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::insert**

Пример использования метода insert() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset;

std::set<int>::iterator it;

std::pair<std::set<int>::iterator, bool> ret;

// установите некоторые начальные значения:

for (int i = 1; i <= 5; ++i) myset.insert(i \* 10); // набор: 10 20 30 40 50

ret = myset.insert(20); // новый элемент не вставлен

if (ret.second == false) it = ret.first; // "it" теперь указывает на элемент 20

myset.insert(it, 25); // максимальная эффективность вставки

myset.insert(it, 24); // максимальная эффективность вставки

myset.insert(it, 26); // не максимальная эффективность вставки

int myints[] = { 5,10,15 }; // 10 уже установлено, не вставлено

myset.insert(myints, myints + 3);

std::cout << "myset contains:";

for (it = myset.begin(); it != myset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::erase**

Пример использования метода erase() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset;

std::set<int>::iterator it;

// вставить некоторые значения:

for (int i = 1; i < 10; i++) myset.insert(i \* 10); // 10 20 30 40 50 60 70 80 90

it = myset.begin();

++it; // "it" теперь указывает на 20

myset.erase(it);

myset.erase(40);

it = myset.find(60);

myset.erase(it, myset.end());

std::cout << "myset contains:";

for (it = myset.begin(); it != myset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::clear**

Пример использования метода clear() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset;

myset.insert(100);

myset.insert(200);

myset.insert(300);

std::cout << "myset contains:";

for (std::set<int>::iterator it = myset.begin(); it != myset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

myset.clear();

myset.insert(1101);

myset.insert(2202);

std::cout << "myset contains:";

for (std::set<int>::iterator it = myset.begin(); it != myset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::swap**

Пример использования метода swap() в std::set:

#include <iostream>

#include <set>

int main() {

int myints[] = { 12,75,10,32,20,25 };

std::set<int> first(myints, myints + 3); // 10,12,75

std::set<int> second(myints + 3, myints + 6); // 20,25,32

first.swap(second);

std::cout << "first contains:";

for (std::set<int>::iterator it = first.begin(); it != first.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

std::cout << "second contains:";

for (std::set<int>::iterator it = second.begin(); it != second.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::count**

Пример использования метода count() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset;

// установите некоторые начальные значения:

for (int i = 1; i < 5; ++i) myset.insert(i \* 3); // набор: 3 6 9 12

for (int i = 0; i < 10; ++i) {

std::cout << i;

if (myset.count(i) != 0)

std::cout << " is an element of myset.\n";

else

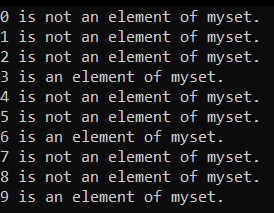
std::cout << " is not an element of myset.\n";

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::find**

Пример использования метода find() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset;

std::set<int>::iterator it;

// установите некоторые начальные значения:

for (int i = 1; i <= 5; i++) myset.insert(i \* 10); // набор: 10 20 30 40 50

it = myset.find(20);

myset.erase(it);

myset.erase(myset.find(40));

std::cout << "myset contains:";

for (it = myset.begin(); it != myset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::equal\_range**

Пример использования метода equal\_range() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset;

for (int i = 1; i <= 5; i++) myset.insert(i \* 10); // myset: 10 20 30 40 50

std::pair<std::set<int>::const\_iterator, std::set<int>::const\_iterator> ret;

ret = myset.equal\_range(30);

std::cout << "the lower bound points to: " << \*ret.first << '\n';

std::cout << "the upper bound points to: " << \*ret.second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::lower\_bound**

Пример использования методов lower\_bound() и upper\_bound() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset;

std::set<int>::iterator itlow, itup;

for (int i = 1; i < 10; i++) myset.insert(i \* 10); // 10 20 30 40 50 60 70 80 90

itlow = myset.lower\_bound(30); // ^

itup = myset.upper\_bound(60); // ^

myset.erase(itlow, itup); // 10 20 70 80 90

std::cout << "myset contains:";

for (std::set<int>::iterator it = myset.begin(); it != myset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Обратите внимание, что lower\_bound(30) возвращает итератор равным 30, тогда как upper\_bound(60) возвращает итератор равным 70.

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::upper\_bound**

#### **set::get\_allocator**

Пример использования метода get\_allocator() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset;

int\* p;

unsigned int i;

// выделите массив из 5 элементов, используя распределитель myset:

p = myset.get\_allocator().allocate(5);

// присвоить массиву некоторые значения

for (i = 0; i < 5; i++) p[i] = (i + 1) \* 10;

std::cout << "The allocated array contains:";

for (i = 0; i < 5; i++) std::cout << ' ' << p[i];

std::cout << '\n';

myset.get\_allocator().deallocate(p, 5);

return 0;

}

В примере показан сложный способ выделения памяти для массива целых чисел с использованием того же распределителя, который используется контейнером.

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::key\_comp**

Пример использования метода key\_comp() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset;

int highest;

std::set<int>::key\_compare mycomp = myset.key\_comp();

for (int i = 0; i <= 5; i++) myset.insert(i);

std::cout << "myset contains:";

highest = \*myset.rbegin();

std::set<int>::iterator it = myset.begin();

do {

std::cout << ' ' << \*it;

} while (mycomp(\*(++it), highest));

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **set::value\_comp**

Пример использования метода value\_comp() в std::set:

#include <iostream>

#include <set>

int main() {

std::set<int> myset;

std::set<int>::value\_compare mycomp = myset.value\_comp();

for (int i = 0; i <= 5; i++) myset.insert(i);

std::cout << "myset contains:";

int highest = \*myset.rbegin();

std::set<int>::iterator it = myset.begin();

do {

std::cout << ' ' << \*it;

} while (mycomp(\*(++it), highest));

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

### **Multiset**

template < class T, // multiset::key\_type/value\_type class Compare = less<T>, // multiet::key\_compare/value\_compare class Alloc = allocator<T> // multiset::allocator\_type > класс multiset;

Мультимножество (multiset) - это контейнер, который хранит элементы в определенном порядке, где несколько элементов могут иметь одинаковые значения.

В мультимножестве значение элемента также служит его идентификатором (значение является ключом типа T). Значение элементов в мультимножестве нельзя изменять (элементы всегда являются константами), но их можно вставлять или удалять из контейнера.

Внутренне элементы в мультимножестве всегда отсортированы в соответствии с определенным критерием строгого частичного порядка, указанным его внутренним объектом сравнения (типа Compare).

Контейнеры multiset обычно медленнее, чем контейнеры unordered\_multiset, при доступе к отдельным элементам по их ключу, но они позволяют прямую итерацию по подмножествам в порядке их следования.

Мультимножества обычно реализуются в виде бинарных деревьев поиска.

**Свойства контейнера**:

* Ассоциативность: элементы в ассоциативных контейнерах обращаются по ключу, а не по абсолютной позиции в контейнере.
* Упорядоченность: элементы в контейнере всегда следуют строгому порядку. Все вставленные элементы получают позицию в этом порядке.
* Множество: значение элемента также является ключом, используемым для его идентификации.
* Множество эквивалентных ключей: в контейнере могут быть несколько элементов с эквивалентными ключами.
* Совместимость с аллокатором: контейнер использует объект аллокатора для динамической работы с памятью.

**Параметры шаблона**:

* T: тип элементов. Каждый элемент в множественном наборе также идентифицируется этим значением (каждое значение является ключом элемента).

Также имеет псевдонимы в виде типов multiset::key\_type и multiset::value\_type.

* Compare: двухаргументный предикат, который принимает два аргумента того же типа, что и элементы, и возвращает значение типа bool. Выражение comp(a,b), где comp - объект этого типа, а a и b - ключевые значения, должно вернуть true, если считается, что a предшествует b в строгом частичном порядке, определенном функцией.

Объект множественного набора использует это выражение для определения порядка следования элементов в контейнере и для сравнения эквивалентности ключей элементов (сравнение выполняется рефлексивно: они эквивалентны, если !comp(a,b) && !comp(b,a)).

Это может быть указатель на функцию или объект функции (см. конструктор для примера). По умолчанию используется less<T>, которая возвращает то же самое, что и оператор "меньше" (a<b).

Также имеет псевдонимы в виде типов multiset::key\_compare и multiset::value\_compare.

* Alloc: тип объекта аллокатора, используемого для определения модели выделения памяти. По умолчанию используется шаблон класса аллокатора, который определяет самую простую модель выделения памяти и не зависит от значения.

Также имеет псевдоним в виде типа multiset::allocator\_type.

#### **multiset::begin**

Пример использования методов begin() и end() в std::multiset:

#include <iostream>

#include <set>

int main() {

int myints[] = { 42,71,71,71,12 };

std::multiset<int> mymultiset(myints, myints + 5);

std::multiset<int>::iterator it;

std::cout << "mymultiset contains:";

for (std::multiset<int>::iterator it = mymultiset.begin(); it != mymultiset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::end**

#### **multiset::rbegin**

Пример использования методов rbegin() и rend() в std::multiset:

#include <iostream>

#include <set>

int main() {

int myints[] = { 77,16,2,30,30 };

std::multiset<int> mymultiset(myints, myints + 5);

std::cout << "mymultiset contains:";

for (std::multiset<int>::reverse\_iterator rit = mymultiset.rbegin(); rit != mymultiset.rend(); ++rit)

std::cout << ' ' << \*rit;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::rend**

#### **multiset::cbegin**

Пример использования методов cbegin() и cend() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset = { 10,20,30,20,10 };

std::cout << "mymultiset contains:";

for (auto it = mymultiset.cbegin(); it != mymultiset.cend(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::cend**

#### **multiset::crbegin**

Пример использования методов crbegin() и crend() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset = { 10,20,30,20,10 };

std::cout << "mymultiset backwards:";

for (auto rit = mymultiset.crbegin(); rit != mymultiset.crend(); ++rit)

std::cout << ' ' << \*rit;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::crend**

#### **multiset::size**

Пример использования метода size() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> myints;

std::cout << "0. size: " << myints.size() << '\n';

for (int i = 0; i < 10; i++) myints.insert(i);

std::cout << "1. size: " << myints.size() << '\n';

myints.insert(5);

std::cout << "2. size: " << myints.size() << '\n';

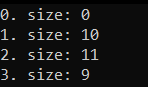
myints.erase(5);

std::cout << "3. size: " << myints.size() << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::max\_size**

Пример использования метода max\_size() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset;

if (mymultiset.max\_size() > 1000) {

for (int i = 0; i < 1000; i++) mymultiset.insert(i);

std::cout << "The multiset contains 1000 elements.\n";

}

else std::cout << "The multiset could not hold 1000 elements.\n";

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::empty**

Пример использования метода empty() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset;

mymultiset.insert(10);

mymultiset.insert(20);

mymultiset.insert(10);

std::cout << "mymultiset contains:";

while (!mymultiset.empty()) {

std::cout << ' ' << \*mymultiset.begin();

mymultiset.erase(mymultiset.begin());

}

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::emplace**

Пример использования метода emplace() в std::multiset:

#include <iostream>

#include <set>

#include <string>

int main() {

std::multiset<std::string> mymultiset;

mymultiset.emplace("foo");

mymultiset.emplace("bar");

mymultiset.emplace("foo");

std::cout << "mymultiset contains:";

for (const std::string& x : mymultiset)

std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::emplace\_hint**

Пример использования метода emplace\_hint() в std::multiset:

#include <iostream>

#include <set>

#include <string>

int main() {

std::multiset<std::string> mymultiset;

auto it = mymultiset.cbegin();

mymultiset.emplace\_hint(it, "apple");

it = mymultiset.emplace\_hint(mymultiset.cend(), "orange");

it = mymultiset.emplace\_hint(it, "melon");

mymultiset.emplace\_hint(it, "melon");

std::cout << "mymultiset contains:";

for (const std::string& x : mymultiset)

std::cout << ' ' << x;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::insert**

Пример использования метода insert() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset;

std::multiset<int>::iterator it;

// установите некоторые начальные значения:

for (int i = 1; i <= 5; i++) mymultiset.insert(i \* 10); // 10 20 30 40 50

it = mymultiset.insert(25);

it = mymultiset.insert(it, 27); // максимальная эффективность вставки

it = mymultiset.insert(it, 29); // максимальная эффективность вставки

it = mymultiset.insert(it, 24); // не максимальная эффективность вставки (24<29)

int myints[] = { 5,10,15 };

mymultiset.insert(myints, myints + 3);

std::cout << "mymultiset contains:";

for (it = mymultiset.begin(); it != mymultiset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::erase**

Пример использования метода erase() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset;

std::multiset<int>::iterator it;

// вставить некоторые значения:

mymultiset.insert(40); // 40

for (int i = 1; i < 7; i++) mymultiset.insert(i \* 10); // 10 20 30 40 40 50 60

it = mymultiset.begin();

it++; // ^

mymultiset.erase(it); // 10 30 40 40 50 60

mymultiset.erase(40); // 10 30 50 60

it = mymultiset.find(50);

mymultiset.erase(it, mymultiset.end()); // 10 30

std::cout << "mymultiset contains:";

for (it = mymultiset.begin(); it != mymultiset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::clear**

Пример использования метода clear() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset;

mymultiset.insert(11);

mymultiset.insert(42);

mymultiset.insert(11);

std::cout << "mymultiset contains:";

for (std::multiset<int>::iterator it = mymultiset.begin(); it != mymultiset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

mymultiset.clear();

mymultiset.insert(200);

mymultiset.insert(100);

std::cout << "mymultiset contains:";

for (std::multiset<int>::iterator it = mymultiset.begin(); it != mymultiset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::swap**

Пример использования метода swap() в std::multiset:

#include <iostream>

#include <set>

int main() {

int myints[] = { 19,72,4,36,20,20 };

std::multiset<int> first(myints, myints + 3); // 4,19,72

std::multiset<int> second(myints + 3, myints + 6); // 20,20,36

first.swap(second);

std::cout << "first contains:";

for (std::multiset<int>::iterator it = first.begin(); it != first.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

std::cout << "second contains:";

for (std::multiset<int>::iterator it = second.begin(); it != second.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::count**

Пример использования метода count() в std::multiset:

#include <iostream>

#include <set>

int main() {

int myints[] = { 10,73,12,22,73,73,12 };

std::multiset<int> mymultiset(myints, myints + 7);

std::cout << "73 appears " << mymultiset.count(73) << " times in mymultiset.\n";

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::find**

Пример использования метода find() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset;

std::multiset<int>::iterator it;

// установите некоторые начальные значения:

for (int i = 1; i <= 5; i++) mymultiset.insert(i \* 10); // 10 20 30 40 50

it = mymultiset.find(20);

mymultiset.erase(it);

mymultiset.erase(mymultiset.find(40));

std::cout << "mymultiset contains:";

for (it = mymultiset.begin(); it != mymultiset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::equal\_range**

Пример использования метода equal\_range() в std::multiset:

#include <iostream>

#include <set>

typedef std::multiset<int>::iterator It;

int main() {

int myints[] = { 77,30,16,2,30,30 };

std::multiset<int> mymultiset(myints, myints + 6); // 2 16 30 30 30 77

std::pair<It, It> ret = mymultiset.equal\_range(30); // ^ ^

mymultiset.erase(ret.first, ret.second);

std::cout << "mymultiset contains:";

for (It it = mymultiset.begin(); it != mymultiset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::lower\_bound**

Пример использования методов lower\_bound() и upper\_bound() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset;

std::multiset<int>::iterator itlow, itup;

for (int i = 1; i < 8; i++) mymultiset.insert(i \* 10); // 10 20 30 40 50 60 70

itlow = mymultiset.lower\_bound(30); // ^

itup = mymultiset.upper\_bound(40); // ^

mymultiset.erase(itlow, itup); // 10 20 50 60 70

std::cout << "mymultiset contains:";

for (std::multiset<int>::iterator it = mymultiset.begin(); it != mymultiset.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

Обратите внимание, что lower\_bound(30) возвращает итератор равным 30, тогда как upper\_bound(40) возвращает итератор равным 50.

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::upper\_bound**

#### **multiset::get\_allocator**

Пример использования метода get\_allocator() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset;

int\* p;

unsigned int i;

// выделите массив из 5 элементов, используя распределитель multiset:

p = mymultiset.get\_allocator().allocate(5);

// присвоить массиву некоторые значения

for (i = 0; i < 5; i++) p[i] = (i + 1) \* 10;

std::cout << "The allocated array contains:";

for (i = 0; i < 5; i++) std::cout << ' ' << p[i];

std::cout << '\n';

mymultiset.get\_allocator().deallocate(p, 5);

return 0;

}

В примере показан сложный способ выделения памяти для массива целых чисел с использованием того же распределителя, который используется контейнером.

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::key\_comp**

Пример использования метода key\_comp() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset;

for (int i = 0; i < 5; i++) mymultiset.insert(i);

std::multiset<int>::key\_compare mycomp = mymultiset.key\_comp();

std::cout << "mymultiset contains:";

int highest = \*mymultiset.rbegin();

std::multiset<int>::iterator it = mymultiset.begin();

do {

std::cout << ' ' << \*it;

} while (mycomp(\*it++, highest));

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multiset::value\_comp**

Пример использования метода value\_comp() в std::multiset:

#include <iostream>

#include <set>

int main() {

std::multiset<int> mymultiset;

std::multiset<int>::value\_compare mycomp = mymultiset.value\_comp();

for (int i = 0; i < 7; i++) mymultiset.insert(i);

std::cout << "mymultiset contains:";

int highest = \*mymultiset.rbegin();

std::multiset<int>::iterator it = mymultiset.begin();

do {

std::cout << ' ' << \*it;

} while (mycomp(\*it++, highest));

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

## **Map**

### **Map**

template < class Key, // map::key\_type

class T, // map::mapped\_type

class Compare = less<Key>, // map::key\_compare

class Alloc = allocator<pair<const Key,T>> //map::allocator\_type > класс map;

Класс map представляет собой ассоциативный контейнер, который хранит элементы в виде комбинации ключа и значения, с определенным порядком следования.

В map ключи используются для сортировки и уникальной идентификации элементов, а значения хранят содержимое, связанное с этим ключом. Типы ключей и значений могут отличаться и объединены в тип "value\_type", который представляет собой пару значений:



Внутри map элементы всегда отсортированы по ключу в соответствии с заданным порядком, определенным объектом сравнения (тип "Compare").

Контейнеры типа map обычно медленнее доступа к отдельным элементам по ключу по сравнению с контейнерами unordered\_map, но позволяют прямую итерацию по подмножеству элементов в порядке их следования.

Значения в map можно получить по соответствующему ключу с помощью оператора квадратных скобок ((operator[]).

map обычно реализуется в виде двоичного дерева поиска.

**Свойства контейнера**:

* Ассоциативный: элементы в ассоциативных контейнерах ссылается по ключу, а не по абсолютной позиции в контейнере.
* Упорядоченный: элементы в контейнере всегда следуют строгому порядку. Все вставленные элементы имеют свою позицию в этом порядке.
* Map: каждый элемент ассоциирует ключ с значением. Ключи служат для идентификации элементов, а значения представляют собой основное содержимое элементов.
* Уникальные ключи: в контейнере не может быть двух элементов с эквивалентными ключами.
* Аллокатор: контейнер использует аллокатор для управления памятью.

**Параметры шаблона**:

* Key: тип ключей. Каждый элемент в map уникально идентифицируется своим ключом.
* T: тип значения. Каждый элемент в map хранит некоторые данные в качестве значения.
* Compare: бинарный предикат, который принимает два ключа в качестве аргументов и возвращает bool. Этот предикат определяет порядок следования элементов и сравнивает ключи на эквивалентность.
* Alloc: тип аллокатора, используемого для управления выделением памяти.

#### **map::begin**

Пример использования методов begin() и end() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

mymap['b'] = 100;

mymap['a'] = 200;

mymap['c'] = 300;

// показать содержимое:

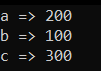
for (std::map<char, int>::iterator it = mymap.begin(); it != mymap.end(); ++it)

std::cout << it->first << " => " << it->second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::end**

#### **map::rbegin**

Пример использования методов rbegin() и rend() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

mymap['x'] = 100;

mymap['y'] = 200;

mymap['z'] = 300;

// показать содержимое:

std::map<char, int>::reverse\_iterator rit;

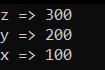
for (rit = mymap.rbegin(); rit != mymap.rend(); ++rit)

std::cout << rit->first << " => " << rit->second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::rend**

#### **map::cbegin**

Пример использования методов cbegin() и cend() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

mymap['b'] = 100;

mymap['a'] = 200;

mymap['c'] = 300;

// вывод:

std::cout << "mymap contains:";

for (auto it = mymap.cbegin(); it != mymap.cend(); ++it)

std::cout << " [" << (\*it).first << ':' << (\*it).second << ']';

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::cend**

#### **map::crbegin**

Пример использования методов crbegin() и crend() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

mymap['b'] = 100;

mymap['a'] = 200;

mymap['c'] = 300;

std::cout << "mymap backwards:";

for (auto rit = mymap.crbegin(); rit != mymap.crend(); ++rit)

std::cout << " [" << rit->first << ':' << rit->second << ']';

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::crend**

#### **map::size**

Пример использования метода size() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

mymap['a'] = 101;

mymap['b'] = 202;

mymap['c'] = 302;

std::cout << "mymap.size() is " << mymap.size() << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::max\_size**

Пример использования метода max\_size() в std::map:

#include <iostream>

#include <map>

int main() {

int i;

std::map<int, int> mymap;

if (mymap.max\_size() > 1000) {

for (i = 0; i < 1000; i++) mymap[i] = 0;

std::cout << "The map contains 1000 elements.\n";

}

else std::cout << "The map could not hold 1000 elements.\n";

return 0;

}

Здесь элемент max\_size используется для предварительной проверки того, позволит ли карта вставить 1000 элементов.

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::empty**

Пример использования метода empty() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

mymap['a'] = 10;

mymap['b'] = 20;

mymap['c'] = 30;

while (!mymap.empty()) {

std::cout << mymap.begin()->first << " => " << mymap.begin()->second << '\n';

mymap.erase(mymap.begin());

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::at**

Пример использования метода at() в std::map:

#include <iostream>

#include <string>

#include <map>

int main() {

std::map<std::string, int> mymap = {

{ "alpha", 0 },

{ "beta", 0 },

{ "gamma", 0 } };

mymap.at("alpha") = 10;

mymap.at("beta") = 20;

mymap.at("gamma") = 30;

for (auto& x : mymap) {

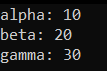
std::cout << x.first << ": " << x.second << '\n';

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::operator[]**

Пример использования метода operarot[] в std::map:

#include <iostream>

#include <map>

#include <string>

int main() {

std::map<char, std::string> mymap;

mymap['a'] = "an element";

mymap['b'] = "another element";

mymap['c'] = mymap['b'];

std::cout << "mymap['a'] is " << mymap['a'] << '\n';

std::cout << "mymap['b'] is " << mymap['b'] << '\n';

std::cout << "mymap['c'] is " << mymap['c'] << '\n';

std::cout << "mymap['d'] is " << mymap['d'] << '\n';

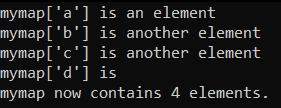
std::cout << "mymap now contains " << mymap.size() << " elements.\n";

return 0;

}

Обратите внимание, как последний доступ (к элементу 'd') вставляет новый элемент в карту с этим ключом и инициализирует его значением по умолчанию (пустой строкой), даже если доступ к нему осуществляется только для извлечения его значения. Функция-член map::find не производит такого эффекта.

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::emplace**

Пример использования метода emplace() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

mymap.emplace('x', 100);

mymap.emplace('y', 200);

mymap.emplace('z', 100);

std::cout << "mymap contains:";

for (auto& x : mymap)

std::cout << " [" << x.first << ':' << x.second << ']';

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::emplace\_hint**

Пример использования метода emplace\_hint() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

auto it = mymap.end();

it = mymap.emplace\_hint(it, 'b', 10);

mymap.emplace\_hint(it, 'a', 12);

mymap.emplace\_hint(mymap.end(), 'c', 14);

std::cout << "mymap contains:";

for (auto& x : mymap)

std::cout << " [" << x.first << ':' << x.second << ']';

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::insert**

Пример использования метода insert() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

// первая версия функции вставки (один параметр):

mymap.insert(std::pair<char, int>('a', 100));

mymap.insert(std::pair<char, int>('z', 200));

std::pair<std::map<char, int>::iterator, bool> ret;

ret = mymap.insert(std::pair<char, int>('z', 500));

if (ret.second == false) {

std::cout << "element 'z' already existed";

std::cout << " with a value of " << ret.first->second << '\n';

}

// вторая версия функции вставки (с положением подсказки):

std::map<char, int>::iterator it = mymap.begin();

mymap.insert(it, std::pair<char, int>('b', 300)); // максимальная эффективность вставки

mymap.insert(it, std::pair<char, int>('c', 400)); // не максимальная эффективность вставки

// третья версия функции вставки (вставка диапазона):

std::map<char, int> anothermap;

anothermap.insert(mymap.begin(), mymap.find('c'));

// отображение содержимого:

std::cout << "mymap contains:\n";

for (it = mymap.begin(); it != mymap.end(); ++it)

std::cout << it->first << " => " << it->second << '\n';

std::cout << "anothermap contains:\n";

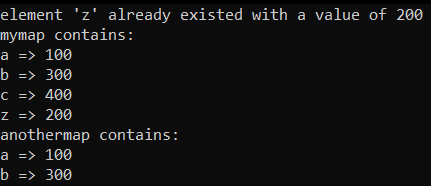
for (it = anothermap.begin(); it != anothermap.end(); ++it)

std::cout << it->first << " => " << it->second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::erase**

Пример использования метода erase() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

std::map<char, int>::iterator it;

// вставить некоторые значения:

mymap['a'] = 10;

mymap['b'] = 20;

mymap['c'] = 30;

mymap['d'] = 40;

mymap['e'] = 50;

mymap['f'] = 60;

it = mymap.find('b');

mymap.erase(it); // стирание с помощью итератора

mymap.erase('c'); // стирание с помощью клавиши

it = mymap.find('e');

mymap.erase(it, mymap.end()); // стирание по диапазону

// показать содержимое:

for (it = mymap.begin(); it != mymap.end(); ++it)

std::cout << it->first << " => " << it->second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::clear**

Пример использования метода clear() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

mymap['x'] = 100;

mymap['y'] = 200;

mymap['z'] = 300;

std::cout << "mymap contains:\n";

for (std::map<char, int>::iterator it = mymap.begin(); it != mymap.end(); ++it)

std::cout << it->first << " => " << it->second << '\n';

mymap.clear();

mymap['a'] = 1101;

mymap['b'] = 2202;

std::cout << "mymap contains:\n";

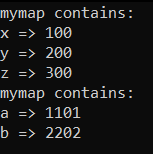
for (std::map<char, int>::iterator it = mymap.begin(); it != mymap.end(); ++it)

std::cout << it->first << " => " << it->second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::swap**

Пример использования метода swap() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> foo, bar;

foo['x'] = 100;

foo['y'] = 200;

bar['a'] = 11;

bar['b'] = 22;

bar['c'] = 33;

foo.swap(bar);

std::cout << "foo contains:\n";

for (std::map<char, int>::iterator it = foo.begin(); it != foo.end(); ++it)

std::cout << it->first << " => " << it->second << '\n';

std::cout << "bar contains:\n";

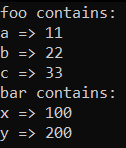
for (std::map<char, int>::iterator it = bar.begin(); it != bar.end(); ++it)

std::cout << it->first << " => " << it->second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::count**

Пример использования метода count() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

char c;

mymap['a'] = 101;

mymap['c'] = 202;

mymap['f'] = 303;

for (c = 'a'; c < 'h'; c++) {

std::cout << c;

if (mymap.count(c) > 0)

std::cout << " is an element of mymap.\n";

else

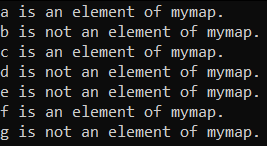
std::cout << " is not an element of mymap.\n";

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::find**

Пример использования метода find() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

std::map<char, int>::iterator it;

mymap['a'] = 50;

mymap['b'] = 100;

mymap['c'] = 150;

mymap['d'] = 200;

it = mymap.find('b');

if (it != mymap.end())

mymap.erase(it);

// вывод:

std::cout << "elements in mymap:" << '\n';

std::cout << "a => " << mymap.find('a')->second << '\n';

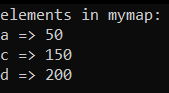
std::cout << "c => " << mymap.find('c')->second << '\n';

std::cout << "d => " << mymap.find('d')->second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::equal\_range**

Пример использования метода equal\_range() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

mymap['a'] = 10;

mymap['b'] = 20;

mymap['c'] = 30;

std::pair<std::map<char, int>::iterator, std::map<char, int>::iterator> ret;

ret = mymap.equal\_range('b');

std::cout << "lower bound points to: ";

std::cout << ret.first->first << " => " << ret.first->second << '\n';

std::cout << "upper bound points to: ";

std::cout << ret.second->first << " => " << ret.second->second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::lower\_bound**

Пример использования методов lower\_bound() и upper\_bound() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

std::map<char, int>::iterator itlow, itup;

mymap['a'] = 20;

mymap['b'] = 40;

mymap['c'] = 60;

mymap['d'] = 80;

mymap['e'] = 100;

itlow = mymap.lower\_bound('b'); // itlow указывает на b

itup = mymap.upper\_bound('d'); // itup указывает на е (не d!)

mymap.erase(itlow, itup); // erases [itlow,itup)

// вывод:

for (std::map<char, int>::iterator it = mymap.begin(); it != mymap.end(); ++it)

std::cout << it->first << " => " << it->second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::upper\_bound**

#### **map::get\_allocator**

Пример использования метода get\_allocator() в std::map:

#include <iostream>

#include <map>

int main() {

int psize;

std::map<char, int> mymap;

std::pair<const char, int>\* p;

// выделите массив из 5 элементов, используя распределитель mymap:

p = mymap.get\_allocator().allocate(5);

// присвоить массиву некоторые значения

psize = sizeof(std::map<char, int>::value\_type) \* 5;

std::cout << "The allocated array has a size of " << psize << " bytes.\n";

mymap.get\_allocator().deallocate(p, 5);

return 0;

}

В примере показан сложный способ выделения памяти для массива пар с использованием того же распределителя, который используется контейнером.

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::key\_comp**

Пример использования метода key\_comp() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

std::map<char, int>::key\_compare mycomp = mymap.key\_comp();

mymap['a'] = 100;

mymap['b'] = 200;

mymap['c'] = 300;

std::cout << "mymap contains:\n";

char highest = mymap.rbegin()->first; // ключевое значение последнего элемента

std::map<char, int>::iterator it = mymap.begin();

do {

std::cout << it->first << " => " << it->second << '\n';

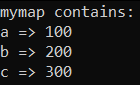
} while (mycomp((\*it++).first, highest));

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **map::value\_comp**

Пример использования метода value\_comp() в std::map:

#include <iostream>

#include <map>

int main() {

std::map<char, int> mymap;

mymap['x'] = 1001;

mymap['y'] = 2002;

mymap['z'] = 3003;

std::cout << "mymap contains:\n";

std::pair<char, int> highest = \*mymap.rbegin(); // последний элемент

std::map<char, int>::iterator it = mymap.begin();

do {

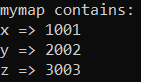
std::cout << it->first << " => " << it->second << '\n';

} while (mymap.value\_comp()(\*it++, highest));

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

### **Multimap**

template < class Key, // multimap::key\_type

class T, // multimap::mapped\_type

class Compare = less<Key>, // multimap::key\_compare

class Alloc = allocator<pair<const Key,T>> // multimap::allocator\_type > класс multimap;

Мультисловарь (multimap) - это ассоциативный контейнер, который хранит элементы, состоящие из комбинации ключевого значения и значения отображения, следуя определенному порядку, где у нескольких элементов могут быть эквивалентные ключи.

В мультисловаре ключевые значения обычно используются для сортировки и уникальной идентификации элементов, в то время как значения отображения хранят связанное с этим ключом содержимое. Типы ключа и значения отображения могут отличаться и объединены в типе значения value\_type, который является парой, объединяющей оба значения:



Внутренне элементы в мультисловаре всегда отсортированы по ключу, следуя определенному критерию строгого частичного порядка, указанному внутренним объектом сравнения (типа Compare).

Контейнеры мультисловарей обычно медленнее, чем контейнеры unordered\_multimap, при доступе к отдельным элементам по их ключу, но они позволяют прямую итерацию по подмножествам в порядке их следования.

Мультисловари обычно реализуются в виде бинарных деревьев поиска.

**Свойства контейнера**:

* Ассоциативный: элементы в ассоциативных контейнерах ссылается по ключу, а не по абсолютной позиции в контейнере.
* Упорядоченный: элементы в контейнере всегда следуют строгому порядку. Все вставленные элементы имеют свою позицию в этом порядке.
* Map: каждый элемент cвязывает ключ с значением. Ключи служат для идентификации элементов, а значения представляют собой основное содержимое элементов.
* Множественные эквивалентные ключи: в контейнере могут существовать несколько элементов с одинаковыми ключами.
* Аллокатор: контейнер использует аллокатор для управления памятью.

**Параметры шаблона**:

* Key: тип ключей элементов. Каждый элемент в мультисловаре идентифицируется своим ключевым значением.
* T: тип значений элементов. Каждый элемент в мультисловаре хранит некоторые данные в качестве значения.
* Compare: бинарный предикат, который принимает два ключа элементов в качестве аргументов и возвращает значение типа bool. Выражение comp(a, b), где comp - объект этого типа, а a и b - ключи элементов, должно возвращать true, если a считается предшествующим b в соответствующем строгом порядке сравнения. По умолчанию используется std::less<Key>, что соответствует применению оператора < для сравнения ключей.
* Alloc: тип объекта аллокатора, используемого для определения модели выделения памяти. По умолчанию используется класс-шаблон аллокатор std::allocator, который определяет наиболее простую модель выделения памяти и не зависит от типа значений элементов.

#### **multimap::begin**

Пример использования методов begin() и end() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap;

mymultimap.insert(std::pair<char, int>('a', 10));

mymultimap.insert(std::pair<char, int>('b', 20));

mymultimap.insert(std::pair<char, int>('b', 150));

// показать содержимое:

for (std::multimap<char, int>::iterator it = mymultimap.begin(); it != mymultimap.end(); ++it)

std::cout << (\*it).first << " => " << (\*it).second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::end**

#### **multimap::rbegin**

Пример использования методов rbegin() и rend() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap;

mymultimap.insert(std::make\_pair('x', 10));

mymultimap.insert(std::make\_pair('y', 20));

mymultimap.insert(std::make\_pair('y', 150));

mymultimap.insert(std::make\_pair('z', 9));

// показать содержимое:

std::multimap<char, int>::reverse\_iterator rit;

for (rit = mymultimap.rbegin(); rit != mymultimap.rend(); ++rit)

std::cout << rit->first << " => " << rit->second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::rend**

#### **multimap::cbegin**

Пример использования методов cbegin() и cend() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap = { {'x',100}, {'y',200}, {'x',300} };

// вывод:

std::cout << "mymultimap contains:";

for (auto it = mymultimap.cbegin(); it != mymultimap.cend(); ++it)

std::cout << " [" << it->first << ':' << it->second << ']';

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::cend**

#### **multimap::crbegin**

Пример использования методов crbegin() и crend() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap = { {'x',100}, {'y',200}, {'x',300} };

// вывод:

std::cout << "mymultimap backwards:";

for (auto rit = mymultimap.crbegin(); rit != mymultimap.crend(); ++rit)

std::cout << " [" << rit->first << ':' << rit->second << ']';

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::crend**

#### **multimap::size**

Пример использования метода size() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap;

mymultimap.insert(std::make\_pair('x', 100));

mymultimap.insert(std::make\_pair('y', 200));

mymultimap.insert(std::make\_pair('y', 350));

mymultimap.insert(std::make\_pair('z', 500));

std::cout << "mymultimap.size() is " << mymultimap.size() << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::max\_size**

Пример использования метода max\_size() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<int, int> mymultimap;

if (mymultimap.max\_size() >= 1000u) {

for (int i = 0; i < 1000; i++)

mymultimap.insert(std::make\_pair(i, 0));

std::cout << "The multimap contains 1000 elements.\n";

}

else std::cout << "The multimap could not hold 1000 elements.\n";

return 0;

}

Здесь элемент max\_size используется для предварительной проверки того, позволит ли multimap вставить 1000 элементов.

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::empty**

Пример использования метода empty() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap;

mymultimap.insert(std::pair<char, int>('b', 101));

mymultimap.insert(std::pair<char, int>('b', 202));

mymultimap.insert(std::pair<char, int>('q', 505));

while (!mymultimap.empty()) {

std::cout << mymultimap.begin()->first << " => ";

std::cout << mymultimap.begin()->second << '\n';

mymultimap.erase(mymultimap.begin());

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::emplace**

Пример использования метода emplace() в std::multimap:

#include <iostream>

#include <string>

#include <map>

int main() {

std::multimap<std::string, float> mymultimap;

mymultimap.emplace("apple", 1.50);

mymultimap.emplace("coffee", 2.10);

mymultimap.emplace("apple", 1.40);

std::cout << "mymultimap contains:";

for (auto& x : mymultimap)

std::cout << " [" << x.first << ':' << x.second << ']';

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::emplace\_hint**

Пример использования метода emplace\_hint() в std::multimap:

#include <iostream>

#include <string>

#include <map>

int main() {

std::multimap<std::string, int> mymultimap;

auto it = mymultimap.end();

it = mymultimap.emplace\_hint(it, "foo", 10);

mymultimap.emplace\_hint(it, "bar", 20);

mymultimap.emplace\_hint(mymultimap.end(), "foo", 30);

std::cout << "mymultimap contains:";

for (auto& x : mymultimap)

std::cout << " [" << x.first << ':' << x.second << ']';

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::insert**

Пример использования метода insert() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap;

std::multimap<char, int>::iterator it;

// первая версия функции вставки (один параметр):

mymultimap.insert(std::pair<char, int>('a', 100));

mymultimap.insert(std::pair<char, int>('z', 150));

it = mymultimap.insert(std::pair<char, int>('b', 75));

// вторая версия функции вставки (с положением подсказки):

mymultimap.insert(it, std::pair<char, int>('c', 300)); // максимальная эффективность вставки

mymultimap.insert(it, std::pair<char, int>('z', 400)); // не максимальная эффективность вставки

// третья версия функции вставки (вставка диапазона):

std::multimap<char, int> anothermultimap;

anothermultimap.insert(mymultimap.begin(), mymultimap.find('c'));

// отображение содержимого:

std::cout << "mymultimap contains:\n";

for (it = mymultimap.begin(); it != mymultimap.end(); ++it)

std::cout << (\*it).first << " => " << (\*it).second << '\n';

std::cout << "anothermultimap contains:\n";

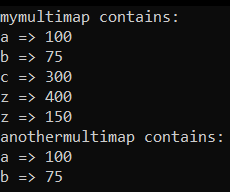
for (it = anothermultimap.begin(); it != anothermultimap.end(); ++it)

std::cout << (\*it).first << " => " << (\*it).second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::erase**

Пример использования метода erase() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap;

// вставить некоторые значения:

mymultimap.insert(std::pair<char, int>('a', 10));

mymultimap.insert(std::pair<char, int>('b', 20));

mymultimap.insert(std::pair<char, int>('b', 30));

mymultimap.insert(std::pair<char, int>('c', 40));

mymultimap.insert(std::pair<char, int>('d', 50));

mymultimap.insert(std::pair<char, int>('d', 60));

mymultimap.insert(std::pair<char, int>('e', 70));

mymultimap.insert(std::pair<char, int>('f', 80));

std::multimap<char, int>::iterator it = mymultimap.find('b');

mymultimap.erase(it); // стирание с помощью итератора (1 элемент)

mymultimap.erase('d'); // стирание по клавише (2 элемента)

it = mymultimap.find('e');

mymultimap.erase(it, mymultimap.end()); // стирание по диапазону

// показать содержимое:

for (it = mymultimap.begin(); it != mymultimap.end(); ++it)

std::cout << (\*it).first << " => " << (\*it).second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::clear**

Пример использования метода clear() в std::multimap:

# #include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap;

std::multimap<char, int>::iterator it;

mymultimap.insert(std::pair<char, int>('b', 80));

mymultimap.insert(std::pair<char, int>('b', 120));

mymultimap.insert(std::pair<char, int>('q', 360));

std::cout << "mymultimap contains:\n";

for (it = mymultimap.begin(); it != mymultimap.end(); ++it)

std::cout << (\*it).first << " => " << (\*it).second << '\n';

mymultimap.clear();

mymultimap.insert(std::pair<char, int>('a', 11));

mymultimap.insert(std::pair<char, int>('x', 22));

std::cout << "mymultimap contains:\n";

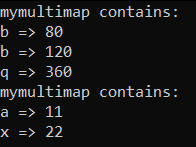
for (it = mymultimap.begin(); it != mymultimap.end(); ++it)

std::cout << (\*it).first << " => " << (\*it).second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::swap**

Пример использования метода swap() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> foo, bar;

foo.insert(std::make\_pair('x', 100));

foo.insert(std::make\_pair('y', 200));

bar.insert(std::make\_pair('a', 11));

bar.insert(std::make\_pair('b', 22));

bar.insert(std::make\_pair('a', 55));

foo.swap(bar);

std::cout << "foo contains:\n";

for (std::multimap<char, int>::iterator it = foo.begin(); it != foo.end(); ++it)

std::cout << (\*it).first << " => " << (\*it).second << '\n';

std::cout << "bar contains:\n";

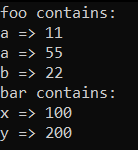
for (std::multimap<char, int>::iterator it = bar.begin(); it != bar.end(); ++it)

std::cout << (\*it).first << " => " << (\*it).second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::count**

Пример использования метода count() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymm;

mymm.insert(std::make\_pair('x', 50));

mymm.insert(std::make\_pair('y', 100));

mymm.insert(std::make\_pair('y', 150));

mymm.insert(std::make\_pair('y', 200));

mymm.insert(std::make\_pair('z', 250));

mymm.insert(std::make\_pair('z', 300));

for (char c = 'x'; c <= 'z'; c++) {

std::cout << "There are " << mymm.count(c) << " elements with key " << c << ":";

std::multimap<char, int>::iterator it;

for (it = mymm.equal\_range(c).first; it != mymm.equal\_range(c).second; ++it)

std::cout << ' ' << (\*it).second;

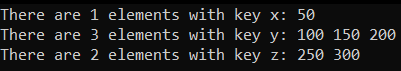
std::cout << '\n';

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::find**

Пример использования метода find() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymm;

mymm.insert(std::make\_pair('x', 10));

mymm.insert(std::make\_pair('y', 20));

mymm.insert(std::make\_pair('z', 30));

mymm.insert(std::make\_pair('z', 40));

std::multimap<char, int>::iterator it = mymm.find('x');

mymm.erase(it);

mymm.erase(mymm.find('z'));

// вывод:

std::cout << "elements in mymm:" << '\n';

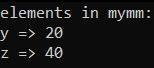
std::cout << "y => " << mymm.find('y')->second << '\n';

std::cout << "z => " << mymm.find('z')->second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::equal\_range**

Пример использования метода equal\_range() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymm;

mymm.insert(std::pair<char, int>('a', 10));

mymm.insert(std::pair<char, int>('b', 20));

mymm.insert(std::pair<char, int>('b', 30));

mymm.insert(std::pair<char, int>('b', 40));

mymm.insert(std::pair<char, int>('c', 50));

mymm.insert(std::pair<char, int>('c', 60));

mymm.insert(std::pair<char, int>('d', 60));

std::cout << "mymm contains:\n";

for (char ch = 'a'; ch <= 'd'; ch++) {

std::pair <std::multimap<char, int>::iterator, std::multimap<char, int>::iterator> ret;

ret = mymm.equal\_range(ch);

std::cout << ch << " =>";

for (std::multimap<char, int>::iterator it = ret.first; it != ret.second; ++it)

std::cout << ' ' << it->second;

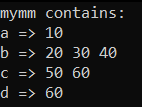
std::cout << '\n';

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::lower\_bound**

Пример использования методов lower\_bound() и upper\_bound() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap;

std::multimap<char, int>::iterator it, itlow, itup;

mymultimap.insert(std::make\_pair('a', 10));

mymultimap.insert(std::make\_pair('b', 121));

mymultimap.insert(std::make\_pair('c', 1001));

mymultimap.insert(std::make\_pair('c', 2002));

mymultimap.insert(std::make\_pair('d', 11011));

mymultimap.insert(std::make\_pair('e', 44));

itlow = mymultimap.lower\_bound('b'); // itlow указывает на b

itup = mymultimap.upper\_bound('d'); // itup указывает на e (не d)

// диапазон печати [itlow,itup):

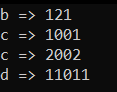
for (it = itlow; it != itup; ++it)

std::cout << (\*it).first << " => " << (\*it).second << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::upper\_bound**

#### **multimap::get\_allocator**

Пример использования метода get\_allocator() в std::multimap:

#include <iostream>

#include <map>

int main() {

int psize;

std::multimap<char, int> mymm;

std::pair<const char, int>\* p;

// выделите массив из 5 элементов с помощью распределителя mymm:

p = mymm.get\_allocator().allocate(5);

// присвоить массиву некоторые значения

psize = sizeof(std::multimap<char, int>::value\_type) \* 5;

std::cout << "The allocated array has a size of " << psize << " bytes.\n";

mymm.get\_allocator().deallocate(p, 5);

return 0;

}

В примере показан сложный способ выделения памяти для массива пар с использованием того же распределителя, который используется контейнером.

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::key\_comp**

Пример использования метода key\_comp() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap;

std::multimap<char, int>::key\_compare mycomp = mymultimap.key\_comp();

mymultimap.insert(std::make\_pair('a', 100));

mymultimap.insert(std::make\_pair('b', 200));

mymultimap.insert(std::make\_pair('b', 211));

mymultimap.insert(std::make\_pair('c', 300));

std::cout << "mymultimap contains:\n";

char highest = mymultimap.rbegin()->first; // ключевое значение последнего элемента

std::multimap<char, int>::iterator it = mymultimap.begin();

do {

std::cout << (\*it).first << " => " << (\*it).second << '\n';

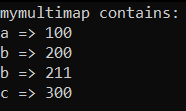
} while (mycomp((\*it++).first, highest));

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

#### **multimap::value\_comp**

Пример использования метода value\_comp() в std::multimap:

#include <iostream>

#include <map>

int main() {

std::multimap<char, int> mymultimap;

mymultimap.insert(std::make\_pair('x', 101));

mymultimap.insert(std::make\_pair('y', 202));

mymultimap.insert(std::make\_pair('y', 252));

mymultimap.insert(std::make\_pair('z', 303));

std::cout << "mymultimap contains:\n";

std::pair<char, int> highest = \*mymultimap.rbegin(); // последний элемент

std::multimap<char, int>::iterator it = mymultimap.begin();

do {

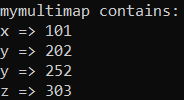
std::cout << (\*it).first << " => " << (\*it).second << '\n';

} while (mymultimap.value\_comp()(\*it++, highest));

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры)

# **Неупорядоченные ассоциативные контейнеры**



## **Unordered\_set**

### **Unordered\_set**

template < class Key, // unordered\_set::key\_type/value\_type class Hash = hash<Key>, // unordered\_set::hasher class Pred = equal\_to<Key>, // unordered\_set::key\_equal class Alloc = allocator<Key> // unordered\_set::allocator\_type > класс unordered\_set;

Неупорядоченное множество (unordered\_set) - это контейнер, который хранит уникальные элементы в произвольном порядке и позволяет быстро находить отдельные элементы по их значению.

В неупорядоченном множестве значение элемента одновременно является его ключом, который идентифицирует его уникально. Ключи неизменяемы, поэтому элементы в неупорядоченном множестве не могут быть изменены после добавления в контейнер, хотя их можно вставлять и удалять.

Внутренне неупорядоченное множество не сортирует элементы в определенном порядке, а организует их в корзины в зависимости от их хеш-значений, чтобы обеспечить быстрый доступ к отдельным элементам напрямую по их значениям (с постоянной средней временной сложностью).

Контейнеры типа unordered\_set обеспечивают более быстрый доступ к отдельным элементам по их ключу, чем контейнеры типа set, хотя они обычно менее эффективны для итерации по диапазону подмножества элементов.

Итераторы в контейнере являются как минимум прямыми итераторами (forward iterators).

**Свойства контейнера**:

* Ассоциативный: элементы в ассоциативных контейнерах определяются и ссылками на их ключи, а не абсолютным положением в контейнере.
* Неупорядоченный: неупорядоченные контейнеры организуют свои элементы с использованием хеш-таблиц, которые позволяют быстро получать доступ к элементам по их ключу.
* Множество: значение элемента также является ключом, используемым для его идентификации.
* Уникальные ключи: в контейнере нет двух элементов с эквивалентными ключами.
* С учетом выделения памяти: контейнер использует объект аллокатора для динамической обработки своих потребностей в памяти.

**Параметры шаблона**:

* Key: тип элементов. Каждый элемент в неупорядоченном множестве также уникально идентифицируется этим значением. Обозначается как ключевой тип (key\_type) и тип значения (value\_type) в неупорядоченном множестве.
* Hash: тип объекта функции, которая принимает объект того же типа, что и элементы, в качестве аргумента и возвращает уникальное значение типа size\_t на основе этого объекта. Это может быть класс, реализующий оператор вызова функции, или указатель на функцию (см. конструктор для примера). По умолчанию используется хеш-функция hash<Key>, которая возвращает значение хеша с вероятностью коллизий, стремящейся к 1,0/std::numeric\_limits<size\_t>::max().

Объект неупорядоченного множества использует возвращаемые этой функцией хеш-значения для внутренней организации своих элементов, ускоряя процесс нахождения отдельных элементов.

Обозначается как хеш-функция (hasher) в неупорядоченном множестве.

* Pred: двоичный предикат, который принимает два аргумента того же типа, что и элементы, и возвращает значение bool. Выражение pred(a,b), где pred - объект этого типа, а a и b - ключевые значения, должно возвращать true, если a должно считаться эквивалентным b. Это может быть класс, реализующий оператор вызова функции, или указатель на функцию (см. конструктор для примера). По умолчанию используется предикат equal\_to<Key>, который возвращает то же значение, что и применение оператора равенства (a == b).

Объект неупорядоченного множества использует это выражение для определения эквивалентности двух ключей элементов. В неупорядоченном множестве нет двух элементов, у которых ключи дают истинное значение при использовании этого предиката.

Обозначается как предикат эквивалентности ключей (key\_equal) в неупорядоченном множестве.

* Alloc: тип объекта аллокатора, используемого для определения модели выделения памяти. По умолчанию используется шаблон класса аллокатора, который определяет простейшую модель выделения памяти и не зависит от значения.

Обозначается как тип аллокатора (allocator\_type) в неупорядоченном множестве.

#### **unordered\_set::begin**

Пример использования методов begin() и end() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset =

{ "Mercury","Venus","Earth","Mars","Jupiter","Saturn","Uranus","Neptune" };

std::cout << "myset contains:";

for (auto it = myset.begin(); it != myset.end(); ++it)

std::cout << " " << \*it;

std::cout << std::endl;

std::cout << "myset's buckets contain:\n";

for (unsigned i = 0; i < myset.bucket\_count(); ++i) {

std::cout << "bucket #" << i << " contains:";

for (auto local\_it = myset.begin(i); local\_it != myset.end(i); ++local\_it)

std::cout << " " << \*local\_it;

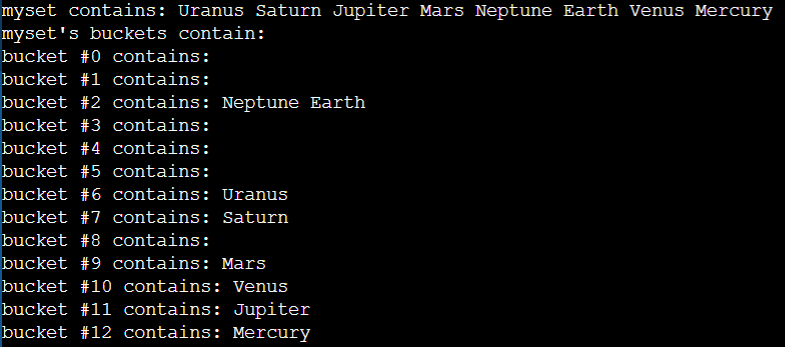
std::cout << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::end**

#### **unordered\_set::cbegin**

Пример использования методов cbegin() и cend() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset =

{ "Mercury","Venus","Earth","Mars","Jupiter","Saturn","Uranus","Neptune" };

std::cout << "myset contains:";

for (auto it = myset.cbegin(); it != myset.cend(); ++it)

std::cout << " " << \*it; // не удается изменить \*it

std::cout << std::endl;

std::cout << "myset's buckets contain:\n";

for (unsigned i = 0; i < myset.bucket\_count(); ++i) {

std::cout << "bucket #" << i << " contains:";

for (auto local\_it = myset.cbegin(i); local\_it != myset.cend(i); ++local\_it)

std::cout << " " << \*local\_it;

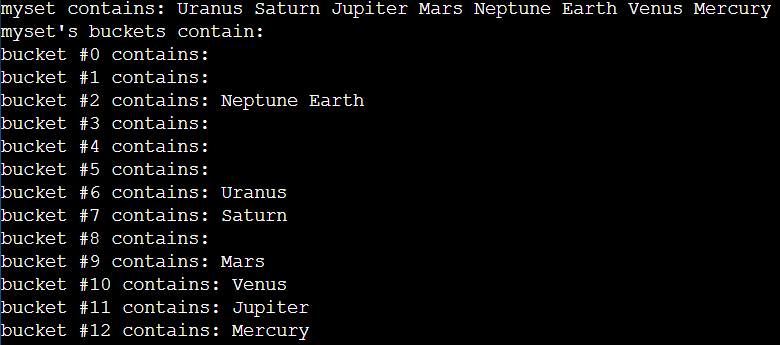
std::cout << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::cend**

#### **unordered\_set::size**

Пример использования метода size() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset;

std::cout << "0. size: " << myset.size() << std::endl;

myset = { "milk","potatoes","eggs" };

std::cout << "1. size: " << myset.size() << std::endl;

myset.insert("pineapple");

std::cout << "2. size: " << myset.size() << std::endl;

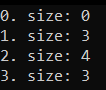
myset.erase("milk");

std::cout << "3. size: " << myset.size() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::max\_size**

Пример использования метода max\_size() в std::unordered\_set:

#include <iostream>

#include <unordered\_set>

int main() {

std::unordered\_set<int> myset;

std::cout << "max\_size = " << myset.max\_size() << std::endl;

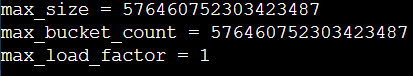
std::cout << "max\_bucket\_count = " << myset.max\_bucket\_count() << std::endl;

std::cout << "max\_load\_factor = " << myset.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::empty**

Пример использования метода empty() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> first;

std::unordered\_set<std::string> second = { "alpha","beta","gamma" };

std::cout << "first " << (first.empty() ? "is empty" : "is not empty") << std::endl;

std::cout << "second " << (second.empty() ? "is empty" : "is not empty") << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::reserve**

Пример использования метода reserve() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset;

myset.reserve(5);

myset.insert("office");

myset.insert("house");

myset.insert("gym");

myset.insert("parking");

myset.insert("highway");

std::cout << "myset contains:";

for (const std::string& x : myset) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Вызвав reserve с размером, который мы ожидали для контейнера unordered\_set, мы избежали многократных повторений, которые могли бы возникнуть при увеличении размера контейнера, и оптимизировали размер хэш-таблицы.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::emplace**

Пример использования метода emplace() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset;

myset.emplace("potatoes");

myset.emplace("milk");

myset.emplace("flour");

std::cout << "myset contains:";

for (const std::string& x : myset) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::emplace\_hint**

#### **unordered\_set::insert**

Пример использования метода insert() в std::unordered\_set:

#include <iostream>

#include <string>

#include <array>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset = { "yellow","green","blue" };

std::array<std::string, 2> myarray = { "black","white" };

std::string mystring = "red";

myset.insert(mystring); // вставка копии

myset.insert(mystring + "dish"); // переместить вставку

myset.insert(myarray.begin(), myarray.end()); // ввод диапазона

myset.insert({ "purple","orange" }); // вставка списка инициализаторов

std::cout << "myset contains:";

for (const std::string& x : myset) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::erase**

Пример использования метода erase() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset =

{ "USA","Canada","France","UK","Japan","Germany","Italy" };

myset.erase(myset.begin()); // стирание с помощью итератора

myset.erase("France"); // стирание с помощью клавиши

myset.erase(myset.find("Japan"), myset.end()); // стирание по диапазону

std::cout << "myset contains:";

for (const std::string& x : myset) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::clear**

Пример использования метода clear() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset =

{ "chair", "table", "lamp", "sofa" };

std::cout << "myset contains:";

for (const std::string& x : myset) std::cout << " " << x;

std::cout << std::endl;

myset.clear();

myset.insert("bed");

myset.insert("wardrobe");

myset.insert("nightstand");

std::cout << "myset contains:";

for (const std::string& x : myset) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::swap**

Пример использования метода swap() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string>

first = { "iron","copper","oil" },

second = { "wood","corn","milk" };

first.swap(second);

std::cout << "first:";

for (const std::string& x : first) std::cout << " " << x;

std::cout << std::endl;

std::cout << "second:";

for (const std::string& x : second) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::count**

Пример использования метода count() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset = { "hat", "umbrella", "suit" };

for (auto& x : { "hat","sunglasses","suit","t-shirt" }) {

if (myset.count(x) > 0)

std::cout << "myset has " << x << std::endl;

else

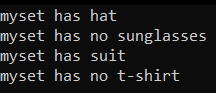
std::cout << "myset has no " << x << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::find**

Пример использования метода find() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset = { "red","green","blue" };

std::string input;

std::cout << "color? ";

getline(std::cin, input);

std::unordered\_set<std::string>::const\_iterator got = myset.find(input);

if (got == myset.end())

std::cout << "not found in myset";

else

std::cout << \*got << " is in myset";

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::equal\_range**

#### **unordered\_set::get\_allocator**

#### **unordered\_set::key\_eq**

Пример использования метода key\_eq() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset;

bool case\_insensitive = myset.key\_eq()("checking", "CHECKING");

std::cout << "myset.key\_eq() is ";

std::cout << (case\_insensitive ? "case insensitive" : "case sensitive");

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::hash\_function**

Пример использования метода hash\_function() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

typedef std::unordered\_set<std::string> stringset;

int main() {

stringset myset;

stringset::hasher fn = myset.hash\_function();

std::cout << "that: " << fn("that") << std::endl;

std::cout << "than: " << fn("than") << std::endl;

return 0;

}

Обратите внимание, что две похожие строки дают совершенно разные хэш-значения.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::bucket**

Пример использования метода bucket() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset = { "water","sand","ice","foam" };

for (const std::string& x : myset) {

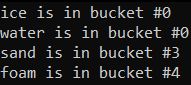
std::cout << x << " is in bucket #" << myset.bucket(x) << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::bucket\_count**

Пример использования метода bucket\_count() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset =

{ "Mercury","Venus","Earth","Mars","Jupiter","Saturn","Uranus","Neptune" };

unsigned n = myset.bucket\_count();

std::cout << "myset has " << n << " buckets.\n";

for (unsigned i = 0; i < n; ++i) {

std::cout << "bucket #" << i << " contains:";

for (auto it = myset.begin(i); it != myset.end(i); ++it)

std::cout << " " << \*it;

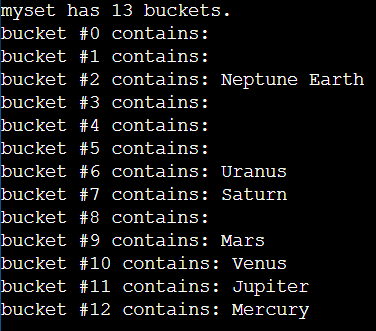
std::cout << "\n";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::bucket\_size**

Пример использования метода bucket\_size() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset =

{ "red", "green", "blue", "yellow", "purple", "pink" };

unsigned nbuckets = myset.bucket\_count();

std::cout << "myset has " << nbuckets << " buckets:\n";

for (unsigned i = 0; i < nbuckets; ++i) {

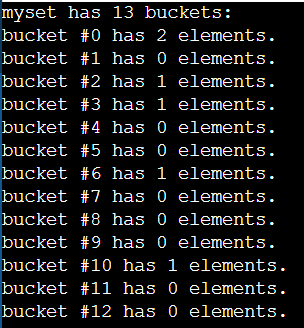
std::cout << "bucket #" << i << " has " << myset.bucket\_size(i) << " elements.\n";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::max\_bucket\_count**

Пример использования метода max\_bucket\_count() в std::unordered\_set:

#include <iostream>

#include <unordered\_set>

int main() {

std::unordered\_set<int> myset;

std::cout << "max\_size = " << myset.max\_size() << std::endl;

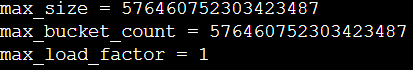
std::cout << "max\_bucket\_count = " << myset.max\_bucket\_count() << std::endl;

std::cout << "max\_load\_factor = " << myset.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::rehash**

Пример использования метода rehash() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset;

myset.rehash(12);

myset.insert("office");

myset.insert("house");

myset.insert("gym");

myset.insert("parking");

myset.insert("highway");

std::cout << "current bucket\_count: " << myset.bucket\_count() << std::endl;

return 0;

}

Вызывая rehash для резервирования определенного минимального количества сегментов в хэш-таблице, мы избегаем многократных повторных хэшей, которые могут возникнуть при расширении контейнера.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::load\_factor**

Пример использования метода load\_factor() в std::unordered\_set:

#include <iostream>

#include <unordered\_set>

int main() {

std::unordered\_set<int> myset;

std::cout << "size = " << myset.size() << std::endl;

std::cout << "bucket\_count = " << myset.bucket\_count() << std::endl;

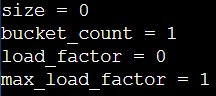
std::cout << "load\_factor = " << myset.load\_factor() << std::endl;

std::cout << "max\_load\_factor = " << myset.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_set::max\_load\_factor**

Пример использования метода max\_load\_factor() в std::unordered\_set:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_set<std::string> myset =

{ "New York", "Paris", "London", "Hong Kong", "Bangalore", "Tel Aviv" };

std::cout << "current max\_load\_factor: " << myset.max\_load\_factor() << std::endl;

std::cout << "current size: " << myset.size() << std::endl;

std::cout << "current bucket\_count: " << myset.bucket\_count() << std::endl;

std::cout << "current load\_factor: " << myset.load\_factor() << std::endl;

float z = myset.max\_load\_factor();

myset.max\_load\_factor(z / 2.0);

std::cout << "[max\_load\_factor halved]" << std::endl;

std::cout << "new max\_load\_factor: " << myset.max\_load\_factor() << std::endl;

std::cout << "new size: " << myset.size() << std::endl;

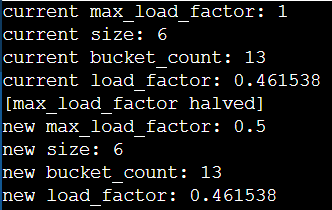
std::cout << "new bucket\_count: " << myset.bucket\_count() << std::endl;

std::cout << "new load\_factor: " << myset.load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

### **Unordered\_multiset**

template < class Key, //unordered\_multiset::key\_type/value\_type class Hash = hash<Key>, // unordered\_multiset::hasher class Pred = equal\_to<Key>, // unordered\_multiset::key\_equal class Alloc = allocator<Key> // unordered\_multiset::allocator\_type > класс unordered\_multiset;

Неупорядоченное мультимножество представляет собой контейнер, в котором элементы хранятся без определенного порядка, позволяя быстро находить отдельные элементы по их значению, подобно контейнерам unordered\_set, но при этом допускается наличие элементов с эквивалентными значениями.

В неупорядоченном мультимножестве значение элемента одновременно является его ключом, используемым для его идентификации. Ключи являются неизменяемыми, поэтому элементы в неупорядоченном мультимножестве не могут быть изменены после помещения в контейнер, но их можно вставлять и удалять.

Внутренне элементы в неупорядоченном мультимножестве не сортируются в определенном порядке, а организуются в корзины в зависимости от их хеш-значений, что позволяет быстро получать доступ к отдельным элементам напрямую по их значениям (с постоянной сложностью времени в среднем).

Элементы с эквивалентными значениями группируются в одну и ту же корзину, таким образом, итератор перебирать все эти элементы.

Итераторы в контейнере являются как минимум прямыми итераторами.

Обратите внимание, что этот контейнер не определен в своем собственном заголовочном файле, а использует заголовочный файл <unordered\_set> вместе с контейнером unordered\_set.

**Свойства контейнера**:

* Ассоциативный: элементы в ассоциативных контейнерах доступны по ключу, а не по их абсолютной позиции в контейнере.
* Неупорядоченный: неупорядоченные контейнеры организуют свои элементы с использованием хеш-таблиц, что позволяет быстро получать доступ к элементам по их ключу.
* Мультимножество: значение элемента также является ключом, используемым для его идентификации.
* Множественные эквивалентные ключи: контейнер может содержать несколько элементов с эквивалентными ключами.
* С поддержкой аллокатора: контейнер использует объект аллокатора для динамического управления своими потребностями в памяти.

**Параметры шаблона**:

* Key: тип элементов. Каждый элемент в неупорядоченном мультимножестве также идентифицируется этим значением.

Типы ключа и значения неупорядоченного мультимножества обозначаются как unordered\_multiset::key\_type и unordered\_multiset::value\_type.

* Hash: унарный функциональный объект, принимающий объект того же типа, что и элементы, в качестве аргумента и возвращающий уникальное значение типа size\_t на основе него. Это может быть либо класс, реализующий оператор вызова функции, либо указатель на функцию (см. конструктор для примера). По умолчанию используется hash<Key>, который возвращает хеш-значение с вероятностью коллизий, стремящейся к 1.0/std::numeric\_limits<size\_t>::max().

Объект unordered\_multiset использует возвращаемые этой функцией хеш-значения для внутренней организации своих элементов, ускоряя процесс нахождения отдельных элементов.

Типы hasher и unordered\_multiset::hasher являются синонимами.

* Pred: бинарный предикат, принимающий два аргумента того же типа, что и элементы, и возвращающий значение типа bool. Выражение pred(a, b), где pred - объект этого типа, а a и b - значения ключей, должно возвращать true, если a считается эквивалентным b. Это может быть либо класс, реализующий оператор вызова функции, либо указатель на функцию (см. конструктор для примера). По умолчанию используется equal\_to<Key>, который возвращает то же значение, что и оператор равенства (a == b).

Объект unordered\_multiset использует это выражение для определения эквивалентности двух ключей элементов. Этот контейнер поддерживает несколько элементов с эквивалентными ключами.

Типы Pred и unordered\_multiset::key\_equal являются синонимами.

* Alloc: тип объекта аллокатора, используемого для определения модели выделения памяти. По умолчанию используется шаблон класса аллокатора, который определяет самую простую модель выделения памяти, независимую от значения.

Тип Alloc и unordered\_multiset::allocator\_type являются синонимами.

#### **unordered\_multiset::begin**

Пример использования методов begin() и end() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums =

{ "father","mother","son","daughter","son","son" };

std::cout << "myums contains:";

for (auto it = myums.begin(); it != myums.end(); ++it)

std::cout << " " << \*it;

std::cout << std::endl;

std::cout << "myums's buckets contain:\n";

for (unsigned i = 0; i < myums.bucket\_count(); ++i) {

std::cout << "bucket #" << i << " contains:";

for (auto local\_it = myums.begin(i); local\_it != myums.end(i); ++local\_it)

std::cout << " " << \*local\_it;

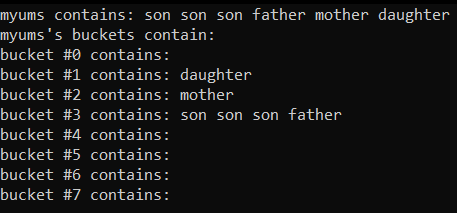
std::cout << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::end**

#### **unordered\_multiset::cbegin**

Пример использования методов cbegin() и cend() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums =

{ "father","mother","son","daughter","son","son" };

std::cout << "myums contains:";

for (auto it = myums.begin(); it != myums.end(); ++it)

std::cout << " " << \*it; // не удается изменить \*it

std::cout << std::endl;

std::cout << "myums's buckets contain:\n";

for (unsigned i = 0; i < myums.bucket\_count(); ++i) {

std::cout << "bucket #" << i << " contains:";

for (auto local\_it = myums.begin(i); local\_it != myums.end(i); ++local\_it)

std::cout << " " << \*local\_it;

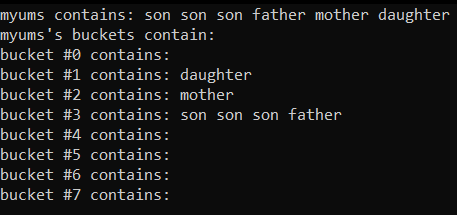
std::cout << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::cend**

#### **unordered\_multiset::size**

Пример использования метода size() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myset;

std::cout << "0. size: " << myset.size() << std::endl;

myset = { "house","parking","office" };

std::cout << "1. size: " << myset.size() << std::endl;

myset.insert("house");

std::cout << "2. size: " << myset.size() << std::endl;

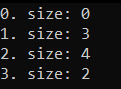
myset.erase("house");

std::cout << "3. size: " << myset.size() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::max\_size**

Пример использования метода max\_size() в std::unordered\_multiset:

#include <iostream>

#include <unordered\_set>

int main() {

std::unordered\_multiset<int> myset;

std::cout << "max\_size = " << myset.max\_size() << std::endl;

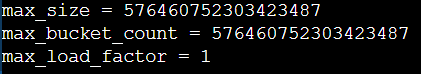
std::cout << "max\_bucket\_count = " << myset.max\_bucket\_count() << std::endl;

std::cout << "max\_load\_factor = " << myset.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::empty**

Пример использования метода empty() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> first;

std::unordered\_multiset<std::string> second = { "mom","dad","son","daughter","son" };

std::cout << "first " << (first.empty() ? "is empty" : "is not empty") << std::endl;

std::cout << "second " << (second.empty() ? "is empty" : "is not empty") << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::reserve**

Пример использования метода reserve() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums;

myums.reserve(5);

myums.insert("klingon");

myums.insert("human");

myums.insert("vulcan");

myums.insert("klingon");

myums.insert("klingon");

std::cout << "myums contains:";

for (const std::string& x : myums) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Вызвав reserve с размером, который мы ожидали для контейнера unordered\_multiset, мы избежали многократных повторений, которые могли бы возникнуть при увеличении размера контейнера, и оптимизировали размер хэш-таблицы.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::emplace**

Пример использования метода emplace() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums;

myums.emplace("milk");

myums.emplace("tea");

myums.emplace("coffee");

myums.emplace("milk");

std::cout << "myums contains:";

for (const std::string& x : myums) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::emplace\_hint**

#### **unordered\_multiset::insert**

Пример использования метода insert() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <array>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums = { "red","green","blue" };

std::array<std::string, 2> myarray = { "red","yellow" };

std::string mystring = "red";

myums.insert(mystring); // вставка копии

myums.insert(mystring + "dish"); // переместить вставку

myums.insert(myarray.begin(), myarray.end()); // ввод диапазона

myums.insert({ "purple","orange" }); // вставка списка инициализаторов

std::cout << "myums contains:";

for (const std::string& x : myums) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::erase**

Пример использования метода erase() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums =

{ "fish","duck","cow","cow","pig","hen","sheep" };

myums.erase(myums.begin()); // стирание с помощью итератора

myums.erase("sheep"); // стирание с помощью клавиши

myums.erase(myums.find("fish"), myums.end()); // стирание по диапазону

std::cout << "myums contains:";

for (const std::string& x : myums) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::clear**

Пример использования метода clear() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums =

{ "chair","table","lamp","sofa","chair" };

std::cout << "myums contains:";

for (const std::string& x : myums) std::cout << " " << x;

std::cout << std::endl;

myums.clear();

myums.insert("bed");

myums.insert("bed");

myums.insert("wardrobe");

myums.insert("nightstand");

std::cout << "myums contains:";

for (const std::string& x : myums) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::swap**

Пример использования метода swap() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string>

first = { "cow","chicken","pig","pig" },

second = { "wolf","rabbit","rabbit" };

first.swap(second);

std::cout << "first:";

for (const std::string& x : first) std::cout << " " << x;

std::cout << std::endl;

std::cout << "second:";

for (const std::string& x : second) std::cout << " " << x;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::count**

Пример использования метода count() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums =

{ "cow","pig","chicken","pig","pig","cow" };

for (auto& x : { "cow","sheep","pig" }) {

std::cout << x << ": " << myums.count(x) << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::find**

Пример использования метода find() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums =

{ "cow","cow","pig","sheep","pig" };

std::unordered\_multiset<std::string>::iterator it = myums.find("pig");

if (it != myums.end())

std::cout << \*it << " found" << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::equal\_range**

Пример использования метода equal\_range() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums =

{ "cow","pig","pig","chicken","pig","chicken" };

auto myrange = myums.equal\_range("pig");

std::cout << "These pigs were found:";

while (myrange.first != myrange.second) {

std::cout << " " << \*myrange.first++;

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::get\_allocator**

#### **unordered\_multiset::key\_eq**

Пример использования метода key\_eq() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums;

bool case\_insensitive = myums.key\_eq()("case", "CASE");

std::cout << "myums.key\_eq() is ";

std::cout << (case\_insensitive ? "case insensitive" : "case sensitive");

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::hash\_function**

Пример использования метода hash\_function() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

typedef std::unordered\_multiset<std::string> stringset;

int main() {

stringset myums;

stringset::hasher fn = myums.hash\_function();

std::cout << "that: " << fn("that") << std::endl;

std::cout << "than: " << fn("than") << std::endl;

return 0;

}

Обратите внимание, что две похожие строки дают совершенно разные хэш-значения.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::bucket**

Пример использования метода bucket() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums =

{ "water","sand","ice","water" };

for (const std::string& x : myums) {

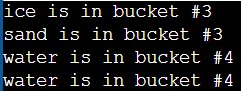
std::cout << x << " is in bucket #" << myums.bucket(x) << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::bucket\_count**

Пример использования метода bucket\_count() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums =

{ "Klingon","Vulcan","Klingon","Cardassian","Vulcan","Human" };

unsigned n = myums.bucket\_count();

std::cout << "myums has " << n << " buckets.\n";

for (unsigned i = 0; i < n; ++i) {

std::cout << "bucket #" << i << " contains:";

for (auto it = myums.begin(i); it != myums.end(i); ++it)

std::cout << " " << \*it;

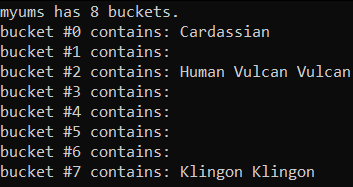
std::cout << "\n";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::bucket\_size**

Пример использования метода bucket\_size() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums =

{ "human","klingon","klingon","vulcan","romulan","cardassian" };

unsigned nbuckets = myums.bucket\_count();

std::cout << "myums has " << nbuckets << " buckets:\n";

for (unsigned i = 0; i < nbuckets; ++i) {

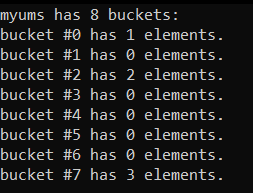
std::cout << "bucket #" << i << " has " << myums.bucket\_size(i) << " elements.\n";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::max\_bucket\_count**

Пример использования метода max\_bucket\_count() в std::unordered\_multiset:

#include <iostream>

#include <unordered\_set>

int main() {

std::unordered\_multiset<int> myums;

std::cout << "max\_size = " << myums.max\_size() << std::endl;

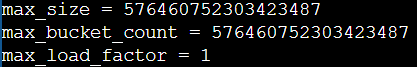
std::cout << "max\_bucket\_count = " << myums.max\_bucket\_count() << std::endl;

std::cout << "max\_load\_factor = " << myums.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::rehash**

Пример использования метода rehash() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums;

myums.rehash(12);

myums.insert("red");

myums.insert("red");

myums.insert("blue");

myums.insert("green");

myums.insert("yellow");

std::cout << "current bucket\_count: " << myums.bucket\_count() << std::endl;

return 0;

}

Вызывая rehash для резервирования определенного минимального количества сегментов в хэш-таблице, мы избегаем многократных повторных хэшей, которые могут возникнуть при расширении контейнера.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::load\_factor**

Пример использования метода load\_factor() в std::unordered\_multiset:

#include <iostream>

#include <unordered\_set>

int main() {

std::unordered\_multiset<int> myums;

std::cout << "size = " << myums.size() << std::endl;

std::cout << "bucket\_count = " << myums.bucket\_count() << std::endl;

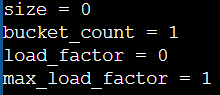
std::cout << "load\_factor = " << myums.load\_factor() << std::endl;

std::cout << "max\_load\_factor = " << myums.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multiset::max\_load\_factor**

Пример использования метода max\_load\_factor() в std::unordered\_multiset:

#include <iostream>

#include <string>

#include <unordered\_set>

int main() {

std::unordered\_multiset<std::string> myums =

{ "human","klingon","vulcan","vulcan","andorian","vulcan" };

std::cout << "current max\_load\_factor: " << myums.max\_load\_factor() << std::endl;

std::cout << "current size: " << myums.size() << std::endl;

std::cout << "current bucket\_count: " << myums.bucket\_count() << std::endl;

std::cout << "current load\_factor: " << myums.load\_factor() << std::endl;

float z = myums.max\_load\_factor();

myums.max\_load\_factor(z / 2.0);

std::cout << "[max\_load\_factor halved]" << std::endl;

std::cout << "new max\_load\_factor: " << myums.max\_load\_factor() << std::endl;

std::cout << "new size: " << myums.size() << std::endl;

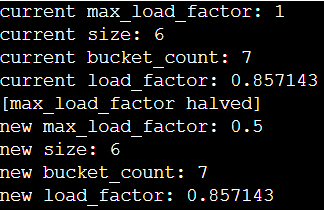
std::cout << "new bucket\_count: " << myums.bucket\_count() << std::endl;

std::cout << "new load\_factor: " << myums.load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

## **Unordered\_map**

### **Unordered\_map**

template < class Key, //unordered\_map::key\_type class T, // unordered\_map::mapped\_type class Hash = hash<Key>, // unordered\_map::hasher class Pred = equal\_to<Key>, // unordered\_map::key\_equal class Alloc = allocator< pair<const Key,T> > // unordered\_map::allocator\_type > класс unordered\_map;

Unordered\_map - это ассоциативный контейнер, который хранит элементы, состоящие из комбинации ключевого значения и значения отображения, и позволяет быстро извлекать отдельные элементы по их ключам.

В unordered\_map ключевое значение обычно используется для уникальной идентификации элемента, в то время как значения отображения представляют объект с связанным содержимым для этого ключа. Типы ключа и значения могут отличаться.

Внутренне элементы в unordered\_map не сортируются по ключу или значению, но они организованы в корзины (buckets) в зависимости от их хэш-значений для обеспечения быстрого доступа к отдельным элементам по их ключам (с постоянной средней сложностью по времени).

Контейнеры unordered\_map обеспечивают более быстрый доступ к отдельным элементам по ключу, чем контейнеры map, хотя они, как правило, менее эффективны для итерации по диапазону подмножества элементов.

Unordered map реализует оператор прямого доступа (operator[]), который позволяет получить прямой доступ к значению отображения, используя ключ в качестве аргумента.

Итераторы в контейнере являются как минимум прямыми итераторами.

**Свойства контейнера**:

* Ассоциативный: элементы в ассоциативных контейнерах доступны по ключу, а не по абсолютной позиции в контейнере.
* Неупорядоченный: неупорядоченные контейнеры организуют свои элементы с использованием хэш-таблиц, что позволяет быстро получать доступ к элементам по ключу.
* Map: каждый элемент связывает ключ с значением. Ключи используются для идентификации элементов, главное содержимое которых - это значение.
* Уникальные ключи: в контейнере не может быть двух элементов с эквивалентными ключами.
* Совместимость с аллокаторами: контейнер использует объект аллокатора для управления выделением памяти.

**Параметры шаблона**:

* Key: тип ключей. Каждый элемент в unordered\_map уникально идентифицируется значением ключа.
* T: тип значений. Каждый элемент в unordered\_map используется для хранения данных в качестве значения.
* Hash: тип объекта-хэш-функции, который принимает объект типа ключа в качестве аргумента и возвращает уникальное значение типа size\_t на основе него. По умолчанию используется hash<Key>, который возвращает хэш-значение с вероятностью коллизии, стремящейся к 1.0/std::numeric\_limits<size\_t>::max().
* Pred: тип бинарного предиката, который принимает два аргумента типа ключа и возвращает bool. По умолчанию используется equal\_to<Key>, который возвращает результат применения оператора равенства (a == b). unordered\_map использует этот предикат для определения эквивалентности ключей элементов.
* Alloc: тип объекта аллокатора, используемого для управления выделением памяти. По умолчанию используется класс-шаблон allocator< pair<const Key, T> >, который определяет наиболее простую модель выделения памяти и не зависит от типа значения.

#### **unordered\_map::begin**

Пример использования методов begin() и end() в std::unordered\_map:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap;

mymap = { {"Australia","Canberra"},{"U.S.","Washington"},{"France","Paris"} };

std::cout << "mymap contains:";

for (auto it = mymap.begin(); it != mymap.end(); ++it)

std::cout << " " << it->first << ":" << it->second;

std::cout << std::endl;

std::cout << "mymap's buckets contain:\n";

for (unsigned i = 0; i < mymap.bucket\_count(); ++i) {

std::cout << "bucket #" << i << " contains:";

for (auto local\_it = mymap.begin(i); local\_it != mymap.end(i); ++local\_it)

std::cout << " " << local\_it->first << ":" << local\_it->second;

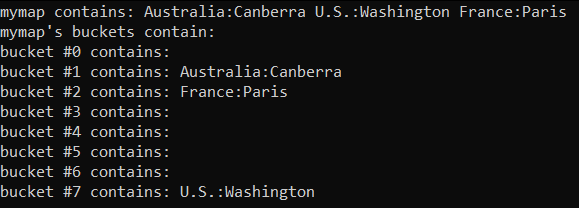
std::cout << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::end**

#### **unordered\_map::cbegin**

Пример использования методов cbegin() и cend() в std::unordered\_map:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap;

mymap = { {"Australia","Canberra"},{"U.S.","Washington"},{"France","Paris"} };

std::cout << "mymap contains:";

for (auto it = mymap.cbegin(); it != mymap.cend(); ++it)

std::cout << " " << it->first << ":" << it->second; // не удается изменить \*it

std::cout << std::endl;

std::cout << "mymap's buckets contain:\n";

for (unsigned i = 0; i < mymap.bucket\_count(); ++i) {

std::cout << "bucket #" << i << " contains:";

for (auto local\_it = mymap.cbegin(i); local\_it != mymap.cend(i); ++local\_it)

std::cout << " " << local\_it->first << ":" << local\_it->second;

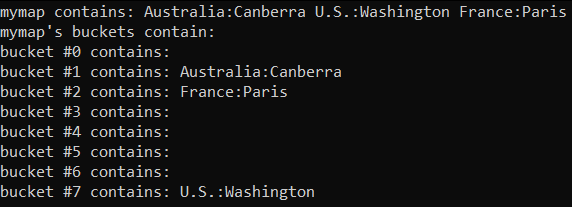
std::cout << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::cend**

#### **unordered\_map::size**

Пример использования метода size() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, double> mymap = {

{"milk",2.30},

{"potatoes",1.90},

{"eggs",0.40}

};

std::cout << "mymap.size() is " << mymap.size() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::max\_size**

Пример использования метода max\_size() в std::unordered\_map:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_map<int, int> mymap;

std::cout << "max\_size = " << mymap.max\_size() << std::endl;

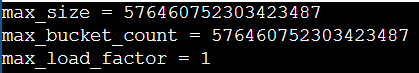
std::cout << "max\_bucket\_count = " << mymap.max\_bucket\_count() << std::endl;

std::cout << "max\_load\_factor = " << mymap.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::empty**

Пример использования метода empty() в std::unordered\_map:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_map<int, int> first;

std::unordered\_map<int, int> second = { {1,10},{2,20},{3,30} };

std::cout << "first " << (first.empty() ? "is empty" : "is not empty") << std::endl;

std::cout << "second " << (second.empty() ? "is empty" : "is not empty") << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::reserve**

Пример использования метода reserve() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap;

mymap.reserve(6);

mymap["house"] = "maison";

mymap["apple"] = "pomme";

mymap["tree"] = "arbre";

mymap["book"] = "livre";

mymap["door"] = "porte";

mymap["grapefruit"] = "pamplemousse";

for (auto& x : mymap) {

std::cout << x.first << ": " << x.second << std::endl;

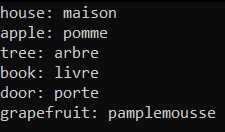
}

return 0;

}

Вызвав reserve с размером, который мы ожидали для контейнера unordered\_map, мы избежали многократных повторений, которые могли бы возникнуть при увеличении размера контейнера, и оптимизировали размер хэш-таблицы.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::at**

Пример использования метода at() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, int> mymap = {

{ "Mars", 3000},

{ "Saturn", 60000},

{ "Jupiter", 70000 } };

mymap.at("Mars") = 3396;

mymap.at("Saturn") += 272;

mymap.at("Jupiter") = mymap.at("Saturn") + 9638;

for (auto& x : mymap) {

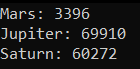
std::cout << x.first << ": " << x.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::operator[]**

Пример использования метода operator[] в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap;

mymap["Bakery"] = "Barbara"; // вставлен новый элемент

mymap["Seafood"] = "Lisa"; // вставлен новый элемент

mymap["Produce"] = "John"; // вставлен новый элемент

std::string name = mymap["Bakery"]; // доступ к существующему элементу (чтение)

mymap["Seafood"] = name; // доступ к существующему элементу (запись)

mymap["Bakery"] = mymap["Produce"]; // доступ к существующим элементам (чтение/запись)

name = mymap["Deli"]; // несуществующий элемент: вставлен новый элемент "Deli"!

mymap["Produce"] = mymap["Gifts"]; // вставлен новый элемент "Gifts", написано "Produce"

for (auto& x : mymap) {

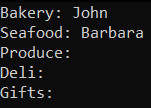
std::cout << x.first << ": " << x.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::emplace**

Пример использования метода emplace() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap;

mymap.emplace("NCC-1701", "J.T. Kirk");

mymap.emplace("NCC-1701-D", "J.L. Picard");

mymap.emplace("NCC-74656", "K. Janeway");

std::cout << "mymap contains:" << std::endl;

for (auto& x : mymap)

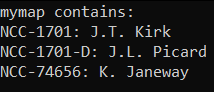
std::cout << x.first << ": " << x.second << std::endl;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::emplace\_hint**

#### **unordered\_map::insert**

Пример использования метода insert() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, double>

myrecipe,

mypantry = { {"milk",2.0},{"flour",1.5} };

std::pair<std::string, double> myshopping("baking powder", 0.3);

myrecipe.insert(myshopping); // вставка копии

myrecipe.insert(std::make\_pair<std::string, double>("eggs", 6.0)); // переместить вставку

myrecipe.insert(mypantry.begin(), mypantry.end()); // ввод диапазона

myrecipe.insert({ {"sugar",0.8},{"salt",0.1} }); // вставка списка инициализаторов

std::cout << "myrecipe contains:" << std::endl;

for (auto& x : myrecipe)

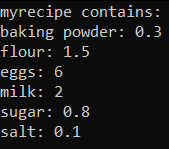
std::cout << x.first << ": " << x.second << std::endl;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::erase**

Пример использования метода erase() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap;

// заполняющий контейнер:

mymap["U.S."] = "Washington";

mymap["U.K."] = "London";

mymap["France"] = "Paris";

mymap["Russia"] = "Moscow";

mymap["China"] = "Beijing";

mymap["Germany"] = "Berlin";

mymap["Japan"] = "Tokyo";

// стереть примеры:

mymap.erase(mymap.begin()); // стирание с помощью итератора

mymap.erase("France"); // стирание с помощью клавиши

mymap.erase(mymap.find("China"), mymap.end()); // стирание по диапазону

// показать содержимое:

for (auto& x : mymap)

std::cout << x.first << ": " << x.second << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::clear**

Пример использования метода clear() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap =

{ {"house","maison"}, {"car","voiture"}, {"grapefruit","pamplemousse"} };

std::cout << "mymap contains:";

for (auto& x : mymap) std::cout << " " << x.first << "=" << x.second;

std::cout << std::endl;

mymap.clear();

mymap["hello"] = "bonjour";

mymap["sun"] = "soleil";

std::cout << "mymap contains:";

for (auto& x : mymap) std::cout << " " << x.first << "=" << x.second;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::swap**

Пример использования метода swap() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string>

first = { {"Star Wars","G. Lucas"},{"Alien","R. Scott"},{"Terminator","J. Cameron"} },

second = { {"Inception","C. Nolan"},{"Donnie Darko","R. Kelly"} };

first.swap(second);

std::cout << "first: ";

for (auto& x : first) std::cout << x.first << " (" << x.second << "), ";

std::cout << std::endl;

std::cout << "second: ";

for (auto& x : second) std::cout << x.first << " (" << x.second << "), ";

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::count**

Пример использования метода count() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, double> mymap = {

{"Burger",2.99},

{"Fries",1.99},

{"Soda",1.50} };

for (auto& x : { "Burger","Pizza","Salad","Soda" }) {

if (mymap.count(x) > 0)

std::cout << "mymap has " << x << std::endl;

else

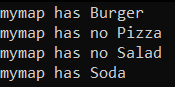
std::cout << "mymap has no " << x << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::find**

Пример использования метода find() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, double> mymap = {

{"mom",5.4},

{"dad",6.1},

{"bro",5.9} };

std::string input;

std::cout << "who? ";

getline(std::cin, input);

std::unordered\_map<std::string, double>::const\_iterator got = mymap.find(input);

if (got == mymap.end())

std::cout << "not found";

else

std::cout << got->first << " is " << got->second;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::equal\_range**

#### **unordered\_map::get\_allocator**

#### **unordered\_map::key\_eq**

Пример использования метода key\_eq() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap;

bool case\_insensitive = mymap.key\_eq()("test", "TEST");

std::cout << "mymap.key\_eq() is ";

std::cout << (case\_insensitive ? "case insensitive" : "case sensitive");

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::hash\_function**

Пример использования метода hash\_function() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

typedef std::unordered\_map<std::string, std::string> stringmap;

int main() {

stringmap mymap;

stringmap::hasher fn = mymap.hash\_function();

std::cout << "this: " << fn("this") << std::endl;

std::cout << "thin: " << fn("thin") << std::endl;

return 0;

}

Обратите внимание, что две похожие строки дают совершенно разные хэш-значения.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::bucket**

Пример использования метода bucket() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap = {

{"us","United States"},

{"uk","United Kingdom"},

{"fr","France"},

{"de","Germany"}

};

for (auto& x : mymap) {

std::cout << "Element [" << x.first << ":" << x.second << "]";

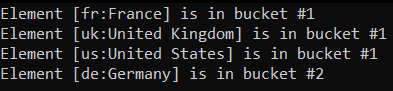
std::cout << " is in bucket #" << mymap.bucket(x.first) << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::bucket\_count**

Пример использования метода bucket\_count() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap = {

{"house","maison"},

{"apple","pomme"},

{"tree","arbre"},

{"book","livre"},

{"door","porte"},

{"grapefruit","pamplemousse"}

};

unsigned n = mymap.bucket\_count();

std::cout << "mymap has " << n << " buckets.\n";

for (unsigned i = 0; i < n; ++i) {

std::cout << "bucket #" << i << " contains: ";

for (auto it = mymap.begin(i); it != mymap.end(i); ++it)

std::cout << "[" << it->first << ":" << it->second << "] ";

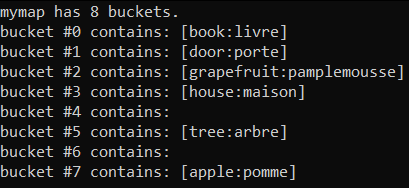
std::cout << "\n";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::bucket\_size**

Пример использования метода bucket\_size() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap = {

{"us","United States"},

{"uk","United Kingdom"},

{"fr","France"},

{"de","Germany"}

};

unsigned nbuckets = mymap.bucket\_count();

std::cout << "mymap has " << nbuckets << " buckets:\n";

for (unsigned i = 0; i < nbuckets; ++i) {

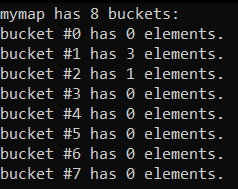
std::cout << "bucket #" << i << " has " << mymap.bucket\_size(i) << " elements.\n";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::max\_bucket\_count**

Пример использования метода max\_bucket\_count() в std::unordered\_map:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_map<int, int> mymap;

std::cout << "max\_size = " << mymap.max\_size() << std::endl;

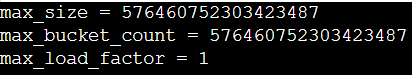
std::cout << "max\_bucket\_count = " << mymap.max\_bucket\_count() << std::endl;

std::cout << "max\_load\_factor = " << mymap.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::rehash**

Пример использования метода rehash() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap;

mymap.rehash(20);

mymap["house"] = "maison";

mymap["apple"] = "pomme";

mymap["tree"] = "arbre";

mymap["book"] = "livre";

mymap["door"] = "porte";

mymap["grapefruit"] = "pamplemousse";

std::cout << "current bucket\_count: " << mymap.bucket\_count() << std::endl;

return 0;

}

Вызывая rehash для резервирования определенного минимального количества сегментов в хэш-таблице, мы избегаем многократных повторных хэшей, которые могут возникнуть при расширении контейнера.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::load\_factor**

Пример использования метода load\_factor() в std::unordered\_map:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_map<int, int> mymap;

std::cout << "size = " << mymap.size() << std::endl;

std::cout << "bucket\_count = " << mymap.bucket\_count() << std::endl;

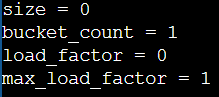
std::cout << "load\_factor = " << mymap.load\_factor() << std::endl;

std::cout << "max\_load\_factor = " << mymap.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_map::max\_load\_factor**

Пример использования метода max\_load\_factor() в std::unordered\_map:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_map<std::string, std::string> mymap = {

{"Au","gold"},

{"Ag","Silver"},

{"Cu","Copper"},

{"Pt","Platinum"}

};

std::cout << "current max\_load\_factor: " << mymap.max\_load\_factor() << std::endl;

std::cout << "current size: " << mymap.size() << std::endl;

std::cout << "current bucket\_count: " << mymap.bucket\_count() << std::endl;

std::cout << "current load\_factor: " << mymap.load\_factor() << std::endl;

float z = mymap.max\_load\_factor();

mymap.max\_load\_factor(z / 2.0);

std::cout << "[max\_load\_factor halved]" << std::endl;

std::cout << "new max\_load\_factor: " << mymap.max\_load\_factor() << std::endl;

std::cout << "new size: " << mymap.size() << std::endl;

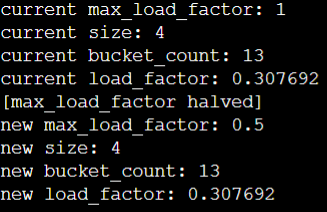
std::cout << "new bucket\_count: " << mymap.bucket\_count() << std::endl;

std::cout << "new load\_factor: " << mymap.load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

### **Unordered\_multimap**

template < class Key, //unordered\_multimap::key\_type class T, // unordered\_multimap::mapped\_type class Hash = hash<Key>, // unordered\_multimap::hasher class Pred = equal\_to<Key>, // unordered\_multimap::key\_equal class Alloc = allocator< pair<const Key,T> //unordered\_multimap::allocator\_type> класс unordered\_multimap;

unordered\_multimap (неупорядоченный мультисловарь) - это ассоциативный контейнер, который хранит элементы, состоящие из комбинации ключа и значения, подобно контейнеру unordered\_map, но с возможностью иметь несколько элементов с одинаковыми ключами.

В unordered\_multimap ключ обычно используется для уникальной идентификации элемента, в то время как значение представляет собой объект, связанный с этим ключом. Типы ключа и значения могут отличаться.

Внутренне элементы в unordered\_multimap не сортируются по ключу или значению, а организуются в корзины (buckets) в зависимости от их хеш-значений, что позволяет быстро получать доступ к отдельным элементам по их ключам (в среднем за постоянное время).

Элементы с эквивалентными ключами группируются в одной корзине, и итератор (см. equal\_range) может перебирать все такие элементы.

Итераторы в контейнере являются как минимум прямыми итераторами.

Обратите внимание, что этот контейнер не определен в своем собственном заголовочном файле, а использует заголовочный файл <unordered\_map> вместе с контейнером unordered\_map.

**Свойства** **контейнера**:

* Ассоциативный: элементы в ассоциативных контейнерах идентифицируются по ключу, а не по абсолютной позиции в контейнере.
* Неупорядоченный: неупорядоченные контейнеры организуют свои элементы с использованием хеш-таблицы, что позволяет быстро получать доступ к элементам по ключу.
* Словарь: каждый элемент ассоциирует ключ с значением. Ключи предназначены для идентификации элементов, основное содержимое которых составляет значение.
* Возможность иметь несколько эквивалентных ключей: контейнер может содержать несколько элементов с эквивалентными ключами.
* Поддержка аллокатора: контейнер использует объект аллокатора для динамической обработки своих потребностей в памяти.

**Параметры шаблона**:

* Key: тип ключевых значений. Каждый элемент в unordered\_multimap идентифицируется значением ключа.
* T: тип значения. Каждый элемент в unordered\_multimap используется для хранения данных в качестве значения.
* Hash: функтор, который принимает объект типа Key в качестве аргумента и возвращает уникальное значение типа size\_t на основе ключа. По умолчанию используется hash<Key>.
* Pred: бинарный предикат, который принимает два аргумента типа Key и возвращает bool. По умолчанию используется equal\_to<Key>.
* Alloc: тип аллокатора, используемого для выделения памяти. По умолчанию используется allocator<pair<const Key, T».

#### **unordered\_multimap::begin**

Пример использования методов begin() и end() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm = {

{"apple","red"},

{"apple","green"},

{"orange","orange"},

{"strawberry","red"}

};

std::cout << "myumm contains:";

for (auto it = myumm.begin(); it != myumm.end(); ++it)

std::cout << " " << it->first << ":" << it->second;

std::cout << std::endl;

std::cout << "myumm's buckets contain:\n";

for (unsigned i = 0; i < myumm.bucket\_count(); ++i) {

std::cout << "bucket #" << i << " contains:";

for (auto local\_it = myumm.begin(i); local\_it != myumm.end(i); ++local\_it)

std::cout << " " << local\_it->first << ":" << local\_it->second;

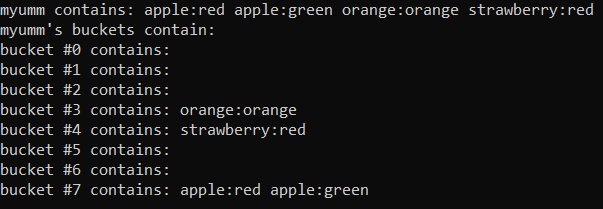
std::cout << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::end**

#### **unordered\_multimap::cbegin**

Пример использования методов cbegin() и cend() в std::unordered\_multimap:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm = {

{"apple","red"},

{"apple","green"},

{"orange","orange"},

{"strawberry","red"}

};

std::cout << "myumm contains:";

for (auto it = myumm.cbegin(); it != myumm.cend(); ++it)

std::cout << " " << it->first << ":" << it->second; // не удается изменить \*it

std::cout << std::endl;

std::cout << "myumm's buckets contain:\n";

for (unsigned i = 0; i < myumm.bucket\_count(); ++i) {

std::cout << "bucket #" << i << " contains:";

for (auto local\_it = myumm.cbegin(i); local\_it != myumm.cend(i); ++local\_it)

std::cout << " " << local\_it->first << ":" << local\_it->second;

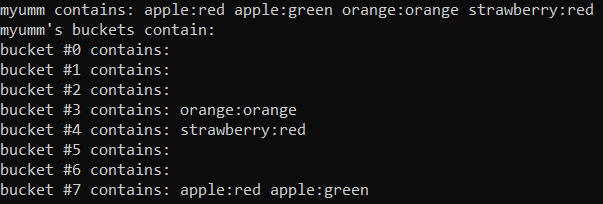
std::cout << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::cend**

#### **unordered\_multimap::size**

Пример использования метода size() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm = {

{"Smith","Pharmacy"},

{"Jones","Library"},

{"Dole","Church"},

{"Smith","Office"}

};

std::cout << "myumm.size() is " << myumm.size() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::max\_size**

Пример использования метода max\_size() в std::unordered\_multimap:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_multimap<int, int> mymap;

std::cout << "max\_size = " << mymap.max\_size() << std::endl;

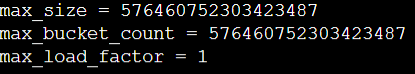
std::cout << "max\_bucket\_count = " << mymap.max\_bucket\_count() << std::endl;

std::cout << "max\_load\_factor = " << mymap.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::empty**

Пример использования метода empty() в std::unordered\_multimap:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_multimap<int, int> first;

std::unordered\_multimap<int, int> second = { {1,10},{2,20},{1,15} };

std::cout << "first " << (first.empty() ? "is empty" : "is not empty") << std::endl;

std::cout << "second " << (second.empty() ? "is empty" : "is not empty") << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::reserve**

Пример использования метода reserve() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm;

myumm.reserve(7);

myumm.insert({ {"apple","NY"},{"apple","WA"},{"peach","GA"} });

myumm.insert({ {"orange","FL"},{"cherry","UT"} });

myumm.insert({ {"strawberry","LA"},{"strawberry","NC"} });

for (auto& x : myumm) {

std::cout << x.first << ": " << x.second << std::endl;

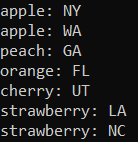
}

return 0;

}

Вызвав reserve с размером, который мы ожидали для контейнера unordered\_multimap, мы избежали многократных повторений, которые могли бы возникнуть при увеличении размера контейнера, и оптимизировали размер хэш-таблицы.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::emplace**

Пример использования метода emplace() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm;

myumm.emplace("NCC-1701", "C. Pike");

myumm.emplace("NCC-1701", "J.T. Kirk");

myumm.emplace("NCC-1701-D", "J.L. Picard");

myumm.emplace("NCC-74656", "K. Janeway");

std::cout << "myumm contains:" << std::endl;

for (auto& x : myumm)

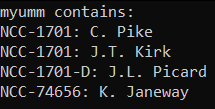
std::cout << x.first << ": " << x.second << std::endl;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::emplace\_hint**

#### **unordered\_multimap::insert**

Пример использования метода insert() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, int>

first,

second = { {"AAPL",200},{"GOOG",100} };

std::pair<std::string, int> mypair("MSFT", 500);

first.insert(mypair); // вставка копии

first.insert(std::make\_pair<std::string, int>("GOOG", 50)); // переместить вставку

first.insert(second.begin(), second.end()); // ввод диапазона

first.insert({ {"ORCL",100},{"GOOG",100} }); // вставка списка инициализаторов

std::cout << "first contains:" << std::endl;

for (auto& x : first)

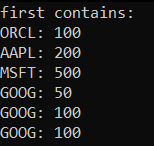
std::cout << x.first << ": " << x.second << std::endl;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::erase**

Пример использования метода erase() в std::unordered\_multimap:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm = {

{"strawberry","red"},

{"banana","yellow"},

{"orange","orange"},

{"lemon","yellow"},

{"apple","red"},

{"apple","green"},

{"pear","green"},

};

// стереть примеры:

myumm.erase(myumm.begin()); // стирание с помощью итератора

myumm.erase("apple"); // стирание с помощью клавиши (стирает 2 элемента)

myumm.erase(myumm.find("orange"), myumm.end()); // стирание по диапазону

// показать содержимое:

for (auto& x : myumm)

std::cout << x.first << ": " << x.second << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::clear**

Пример использования метода clear() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm =

{ {"Tom","Produce"}, {"Bob","Toys"}, {"Bob","Garden"} };

std::cout << "myumm contains:";

for (auto& x : myumm) std::cout << " " << x.first << ":" << x.second;

std::cout << std::endl;

myumm.clear();

myumm.insert(std::make\_pair("Bob", "Jail"));

std::cout << "myumm contains:";

for (auto& x : myumm) std::cout << " " << x.first << ":" << x.second;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::swap**

Пример использования метода swap() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string>

a = { {"orange","FL"},{"apple","NY"},{"apple","WA"} },

b = { {"strawberry","LA"},{"strawberry","NC"},{"pear","OR"} };

a.swap(b);

std::cout << "a: ";

for (auto& x : a) std::cout << " " << x.first << ":" << x.second;

std::cout << std::endl;

std::cout << "b: ";

for (auto& x : b) std::cout << " " << x.first << ":" << x.second;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::count**

Пример использования метода count() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm = {

{"orange","FL"},

{"strawberry","LA"},

{"strawberry","OK"},

{"pumpkin","NH"} };

for (auto& x : { "orange","lemon","strawberry" }) {

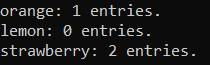
std::cout << x << ": " << myumm.count(x) << " entries.\n";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::find**

Пример использования метода find() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> mymap = {

{"mom","church"},

{"mom","college"},

{"dad","office"},

{"bro","school"} };

std::cout << "one of the values for 'mom' is: ";

std::cout << mymap.find("mom")->second;

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::equal\_range**

Пример использования метода equal\_range() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

#include <algorithm>

typedef std::unordered\_multimap<std::string, std::string> stringmap;

int main() {

stringmap myumm = {

{"orange","FL"},

{"strawberry","LA"},

{"strawberry","OK"},

{"pumpkin","NH"}

};

std::cout << "Entries with strawberry:";

auto range = myumm.equal\_range("strawberry");

for\_each(

range.first,

range.second,

[](stringmap::value\_type& x) {std::cout << " " << x.second;}

);

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::get\_allocator**

#### **unordered\_multimap::key\_eq**

Пример использования метода key\_eq() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm;

bool case\_insensitive = myumm.key\_eq()("test", "TEST");

std::cout << "myumm.key\_eq() is ";

std::cout << (case\_insensitive ? "case insensitive" : "case sensitive");

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::hash\_function**

Пример использования метода hash\_function() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

typedef std::unordered\_multimap<std::string, std::string> stringmap;

int main() {

stringmap myumm;

stringmap::hasher fn = myumm.hash\_function();

std::cout << "this: " << fn("this") << std::endl;

std::cout << "thin: " << fn("thin") << std::endl;

return 0;

}

Обратите внимание, что две похожие строки дают совершенно разные хэш-значения.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::bucket**

Пример использования метода bucket() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm = {

{"John","Middle East"},

{"John","Africa"},

{"Adam","Europe"},

{"Bill","Norh-America"}

};

for (auto& x : myumm) {

std::cout << "Element [" << x.first << ":" << x.second << "]";

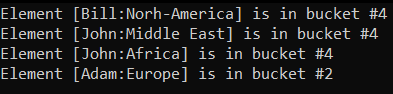
std::cout << " is in bucket #" << myumm.bucket(x.first) << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::bucket\_count**

Пример использования метода bucket\_count() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm = {

{"bed","bedroom"},

{"oven","kitchen"},

{"towel","bathroom"},

{"towel","beach"},

{"plant","garden"}

};

unsigned n = myumm.bucket\_count();

std::cout << "myumm has " << n << " buckets.\n";

for (unsigned i = 0; i < n; ++i) {

std::cout << "bucket #" << i << " contains: ";

for (auto it = myumm.begin(i); it != myumm.end(i); ++it)

std::cout << "[" << it->first << ":" << it->second << "] ";

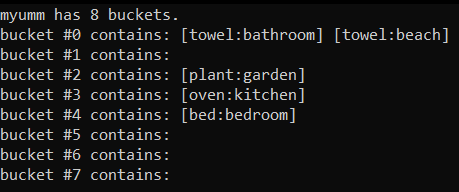
std::cout << "\n";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::bucket\_size**

Пример использования метода bucket\_size() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm = {

{"John","Alpha"},

{"Alfred","Beta"},

{"Thomas","Gamma"},

{"John","Delta"}

};

unsigned nbuckets = myumm.bucket\_count();

std::cout << "myumm has " << nbuckets << " buckets:\n";

for (unsigned i = 0; i < nbuckets; ++i) {

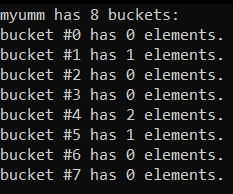
std::cout << "bucket #" << i << " has " << myumm.bucket\_size(i) << " elements.\n";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::max\_bucket\_count**

Пример использования метода max\_bucket\_count() в std::unordered\_multimap:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_multimap<int, int> mymap;

std::cout << "max\_size = " << mymap.max\_size() << std::endl;

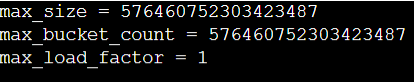
std::cout << "max\_bucket\_count = " << mymap.max\_bucket\_count() << std::endl;

std::cout << "max\_load\_factor = " << mymap.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::rehash**

Пример использования метода rehash() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm;

myumm.rehash(20);

myumm.insert({ {"apple","NY"},{"apple","WA"},{"peach","GA"} });

myumm.insert({ {"orange","FL"},{"cherry","UT"} });

myumm.insert({ {"strawberry","LA"},{"strawberry","NC"} });

std::cout << "current bucket\_count: " << myumm.bucket\_count() << std::endl;

return 0;

}

Вызывая rehash для резервирования определенного минимального количества сегментов в хэш-таблице, мы избегаем многократных повторных хэшей, которые могут возникнуть при расширении контейнера.

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::load\_factor**

Пример использования метода load\_factor() в std::unordered\_multimap:

#include <iostream>

#include <unordered\_map>

int main() {

std::unordered\_multimap<int, int> mymap;

std::cout << "size = " << mymap.size() << std::endl;

std::cout << "bucket\_count = " << mymap.bucket\_count() << std::endl;

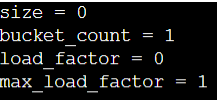
std::cout << "load\_factor = " << mymap.load\_factor() << std::endl;

std::cout << "max\_load\_factor = " << mymap.max\_load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

#### **unordered\_multimap::max\_load\_factor**

Пример использования метода max\_load\_factor() в std::unordered\_multimap:

#include <iostream>

#include <string>

#include <unordered\_map>

int main() {

std::unordered\_multimap<std::string, std::string> myumm = {

{"apple","NY"},

{"apple","WA"},

{"peach","GA"},

{"cherry","UT"}

};

std::cout << "current max\_load\_factor: " << myumm.max\_load\_factor() << std::endl;

std::cout << "current size: " << myumm.size() << std::endl;

std::cout << "current bucket\_count: " << myumm.bucket\_count() << std::endl;

std::cout << "current load\_factor: " << myumm.load\_factor() << std::endl;

float z = myumm.max\_load\_factor();

myumm.max\_load\_factor(z / 2.0);

std::cout << "[max\_load\_factor halved]" << std::endl;

std::cout << "new max\_load\_factor: " << myumm.max\_load\_factor() << std::endl;

std::cout << "new size: " << myumm.size() << std::endl;

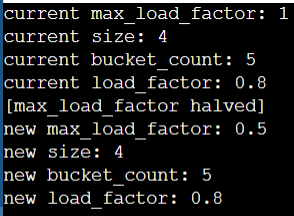
std::cout << "new bucket\_count: " << myumm.bucket\_count() << std::endl;

std::cout << "new load\_factor: " << myumm.load\_factor() << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Ассоциативные_контейнеры_1)

# **Адаптеры для контейнеров**

## **Stack**

template <class T, class Container = deque<T> > класс stack;

Стеки являются контейнерными адаптерами, специально разработанными для работы в контексте LIFO (last-in first-out), где элементы вставляются и извлекаются только с одного конца контейнера.

Стеки реализуются в виде контейнерных адаптеров, которые используют внутренний контейнер определенного типа в качестве базового контейнера, предоставляя набор функций-членов для доступа к его элементам. Элементы добавляются и удаляются с "заднего" конца конкретного контейнера, который называется вершиной стека.

В качестве базового контейнера может использоваться один из стандартных шаблонов классов контейнеров или другой специально разработанный класс контейнера. Этот базовый контейнер должен поддерживать следующие операции:

* empty(): проверяет, является ли стек пустым
* size(): возвращает количество элементов в стеке
* back(): получает доступ к верхнему элементу стека
* push\_back(): добавляет элемент в верхушку стека
* pop\_back(): удаляет верхний элемент стека

Стандартные классы контейнеров vector, deque и list удовлетворяют этим требованиям. По умолчанию, если не указан класс контейнера при создании объекта класса stack, используется стандартный контейнер deque.

**Параметры шаблона**:

* T: тип элементов в стеке.
* Container: тип внутреннего контейнера, в котором хранятся элементы.

Класс std::stack также определяет несколько типов-членов, включая value\_type, container\_type и size\_type, которые представляют типы элементов, тип внутреннего контейнера и беззнаковый целочисленный тип (обычно size\_t) соответственно.

Класс std::stack предоставляет ряд функций-членов, включая конструкторы, функции для проверки пустоты, получения размера, доступа к верхнему элементу, вставки элемента, конструирования и вставки элемента, удаления верхнего элемента и обмена содержимым с другим стеком.

Также имеются перегрузки не-членовых функций, включая операторы отношения и функцию swap, позволяющую обменять содержимое между двумя стеками.

#### **stack::empty**

Пример использования метода empty() в std::stack:

#include <iostream>

#include <stack>

int main() {

std::stack<int> mystack;

int sum(0);

for (int i = 1;i <= 10;i++) mystack.push(i);

while (!mystack.empty()) {

sum += mystack.top();

mystack.pop();

}

std::cout << "total: " << sum << '\n';

return 0;

}

В примере содержимое стека инициализируется последовательностью чисел (от 1 до 10). Затем он извлекает элементы один за другим, пока не станет пустым, и вычисляет их сумму.

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **stack::size**

Пример использования метода size() в std::stack:

#include <iostream>

#include <stack>

int main() {

std::stack<int> myints;

std::cout << "0. size: " << myints.size() << '\n';

for (int i = 0; i < 5; i++) myints.push(i);

std::cout << "1. size: " << myints.size() << '\n';

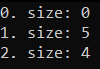
myints.pop();

std::cout << "2. size: " << myints.size() << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **stack::top**

Пример использования метода top() в std::stack:

#include <iostream>

#include <stack>

int main() {

std::stack<int> mystack;

mystack.push(10);

mystack.push(20);

mystack.top() -= 5;

std::cout << "mystack.top() is now " << mystack.top() << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **stack::push**

Пример использования методов push() и pop() в std::stack:

#include <iostream>

#include <stack>

int main() {

std::stack<int> mystack;

for (int i = 0; i < 5; ++i) mystack.push(i);

std::cout << "Popping out elements...";

while (!mystack.empty()) {

std::cout << ' ' << mystack.top();

mystack.pop();

}

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **stack::emplace**

Пример использования метода emplace() в std::stack:

#include <iostream>

#include <stack>

#include <string>

int main() {

std::stack<std::string> mystack;

mystack.emplace("First sentence");

mystack.emplace("Second sentence");

std::cout << "mystack contains:\n";

while (!mystack.empty()) {

std::cout << mystack.top() << '\n';

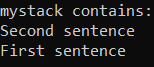
mystack.pop();

}

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **stack::pop**

#### **stack::swap**

Пример использования метода swap() в std::stack:

#include <iostream>

#include <stack>

int main() {

std::stack<int> foo, bar;

foo.push(10); foo.push(20); foo.push(30);

bar.push(111); bar.push(222);

foo.swap(bar);

std::cout << "size of foo: " << foo.size() << '\n';

std::cout << "size of bar: " << bar.size() << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

## **Queue**

template <class T, class Container = deque<T> > класс queue;

Очереди являются контейнерными адаптерами, специально разработанными для работы в контексте FIFO (первым пришел - первым вышел), где элементы вставляются в один конец контейнера и извлекаются с другого.

Очереди реализуются в виде контейнерных адаптеров, которые используют внутренний контейнер определенного типа в качестве базового контейнера, предоставляя набор функций-членов для доступа к его элементам. Элементы добавляются в "конец" конкретного контейнера и извлекаются из его "начала".

В качестве базового контейнера может использоваться один из стандартных шаблонов классов контейнеров или другой специально разработанный класс контейнера. Этот базовый контейнер должен поддерживать как минимум следующие операции:

* empty(): проверяет, является ли очередь пустой
* size(): возвращает количество элементов в очереди
* front(): получает доступ к следующему элементу в очереди
* back(): получает доступ к последнему элементу в очереди
* push\_back(): добавляет элемент в конец очереди
* pop\_front(): удаляет следующий элемент из очереди

Стандартные классы контейнеров deque и list удовлетворяют этим требованиям. По умолчанию, если не указан класс контейнера при создании объекта класса queue, используется стандартный контейнер deque.

**Параметры шаблона**:

* T: тип элементов в очереди.
* Container: тип внутреннего контейнера, в котором хранятся элементы.

Класс std::queue также определяет несколько типов-членов, включая value\_type, container\_type и size\_type, которые представляют типы элементов, тип внутреннего контейнера и беззнаковый целочисленный тип (обычно size\_t) соответственно.

Класс std::queue предоставляет ряд функций-членов, включая конструкторы, функции для проверки пустоты, получения размера, доступа к следующему и последнему элементам, вставки элемента, конструирования и вставки элемента, удаления следующего элемента и обмена содержимым с другой очередью.

Также имеются перегрузки не-членовых функций, включая операторы отношения и функцию swap, позволяющую обменять содержимое между двумя очередями.

#### **queue::empty**

Пример использования метода empty() в std::queue:

#include <iostream>

#include <queue>

int main() {

std::queue<int> myqueue;

int sum(0);

for (int i = 1;i <= 10;i++) myqueue.push(i);

while (!myqueue.empty()) {

sum += myqueue.front();

myqueue.pop();

}

std::cout << "total: " << sum << '\n';

return 0;

}

В примере содержимое очереди инициализируется последовательностью чисел (от 1 до 10). Затем он извлекает элементы один за другим, пока не станет пустым, и вычисляет их сумму.

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **queue::size**

Пример использования метода size() в std::queue:

#include <iostream>

#include <queue>

int main() {

std::queue<int> myints;

std::cout << "0. size: " << myints.size() << '\n';

for (int i = 0; i < 5; i++) myints.push(i);

std::cout << "1. size: " << myints.size() << '\n';

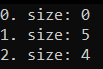
myints.pop();

std::cout << "2. size: " << myints.size() << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **queue::push**

Пример использования методов push() и pop() в std::queue:

#include <iostream>

#include <queue>

int main() {

std::queue<int> myqueue;

int myint;

std::cout << "Please enter some integers (enter 0 to end):\n";

do {

std::cin >> myint;

myqueue.push(myint);

} while (myint);

std::cout << "myqueue contains: ";

while (!myqueue.empty()) {

std::cout << ' ' << myqueue.front();

myqueue.pop();

}

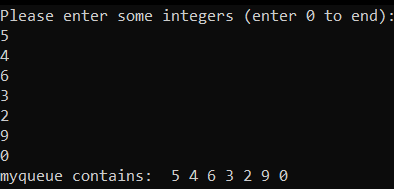
std::cout << '\n';

return 0;

}

В примере используется push для добавления новых элементов в очередь, которые затем выводятся в том же порядке.

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **queue::emplace**

Пример использования метода emplace() в std::queue:

#include <iostream>

#include <queue>

#include <string>

int main() {

std::queue<std::string> myqueue;

myqueue.emplace("First sentence");

myqueue.emplace("Second sentence");

std::cout << "myqueue contains:\n";

while (!myqueue.empty()) {

std::cout << myqueue.front() << '\n';

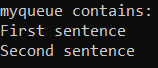
myqueue.pop();

}

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **queue::pop**

#### **queue::swap**

Пример использования метода swap() в std::queue:

#include <iostream>

#include <queue>

int main() {

std::queue<int> foo, bar;

foo.push(10); foo.push(20); foo.push(30);

bar.push(111); bar.push(222);

foo.swap(bar);

std::cout << "size of foo: " << foo.size() << '\n';

std::cout << "size of bar: " << bar.size() << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **queue::front**

Пример использования метода front() в std::queue:

#include <iostream>

#include <queue>

int main() {

std::queue<int> myqueue;

myqueue.push(77);

myqueue.push(16);

myqueue.front() -= myqueue.back();

std::cout << "myqueue.front() is now " << myqueue.front() << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **queue::back**

Пример использования метода back() в std::queue:

#include <iostream>

#include <queue>

int main() {

std::queue<int> myqueue;

myqueue.push(12);

myqueue.push(75); // теперь это back

myqueue.back() -= myqueue.front();

std::cout << "myqueue.back() is now " << myqueue.back() << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

## **Priority\_queue**

template <class T,

class Container = vector<T>,

class Compare = less<typename Container::value\_type> > класс priority\_queue;

Приоритетные очереди являются контейнерными адаптерами, специально разработанными таким образом, чтобы их первый элемент всегда был наибольшим среди элементов, которые они содержат, в соответствии с заданным строгим критерием слабого упорядочивания.

Контекст работы приоритетной очереди похож на кучу (heap), где элементы могут быть вставлены в любой момент, и только максимальный элемент (верхний элемент) может быть извлечен из приоритетной очереди.

Приоритетные очереди реализуются в виде контейнерных адаптеров, которые используют внутренний контейнер определенного типа в качестве базового контейнера, предоставляя набор функций-членов для доступа к элементам. Элементы извлекаются из "задней" части контейнера, которая является верхней частью приоритетной очереди.

В качестве базового контейнера может быть использован любой стандартный контейнер или специально разработанный контейнер. Контейнер должен быть доступен с помощью итераторов произвольного доступа и поддерживать следующие операции:

* empty(): проверяет, является ли очередь пустой
* size(): возвращает количество элементов в очереди
* front(): получает доступ к первому элементу в очереди
* push\_back(): добавляет элемент в конец очереди
* pop\_back(): удаляет последний элемент из очереди

Стандартные контейнеры vector и deque удовлетворяют этим требованиям. По умолчанию, если не указан тип контейнера при создании объекта класса priority\_queue, используется стандартный контейнер vector.

Для поддержки внутренней структуры кучи в приоритетной очереди всегда требуется поддержка итераторов произвольного доступа. Это автоматически обрабатывается контейнерным адаптером, который автоматически вызывает алгоритмические функции make\_heap, push\_heap и pop\_heap при необходимости.

**Параметры шаблона**:

* T: тип элементов в очереди.
* Container: тип внутреннего контейнера, в котором хранятся элементы.
* Compare: бинарный предикат, который принимает два элемента типа T в качестве аргументов и возвращает значение типа bool. Выражение comp(a, b), где comp - объект этого типа, а a и b - элементы в контейнере, должно возвращать true, если a считается предшествующим b в соответствии с заданным строгим упорядочиванием. Приоритетная очередь использует эту функцию для поддержания упорядоченности элементов таким образом, чтобы при извлечении элементов сохранялись свойства кучи (т.е. извлекался последний элемент согласно заданному порядку). По умолчанию используется less<T>, которая возвращает то же самое значение, что и оператор "меньше" (a < b).

Класс std::priority\_queue также определяет несколько типов-членов, включая value\_type, container\_type и size\_type, которые представляют типы элементов, тип внутреннего контейнера и беззнаковый целочисленный тип (обычно size\_t) соответственно.

Класс std::priority\_queue предоставляет ряд функций-членов, включая конструкторы, функции для проверки пустоты, получения размера, доступа к верхнему элементу, вставки элемента, конструирования и вставки элемента, удаления верхнего элемента и обмена содержимым с другой приоритетной очередью.

Кроме того, имеются перегрузки не-членовых функций, включая функцию swap, которая позволяет обменять содержимое между двумя приоритетными очередями.

#### **priority\_queue::empty**

Пример использования метода empty() в std::prioprity\_queue:

#include <iostream>

#include <queue>

int main() {

std::priority\_queue<int> mypq;

int sum(0);

for (int i = 1;i <= 10;i++) mypq.push(i);

while (!mypq.empty()) {

sum += mypq.top();

mypq.pop();

}

std::cout << "total: " << sum << '\n';

return 0;

}

В примере содержимое приоритетной очереди инициализируется последовательностью чисел (от 1 до 10). Затем он извлекает элементы один за другим, пока не станет пустым, и вычисляет их сумму.

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **priority\_queue::size**

Пример использования метода size() в std::prioprity\_queue:

#include <iostream>

#include <queue>

int main() {

std::priority\_queue<int> myints;

std::cout << "0. size: " << myints.size() << '\n';

for (int i = 0; i < 5; i++) myints.push(i);

std::cout << "1. size: " << myints.size() << '\n';

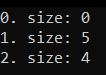
myints.pop();

std::cout << "2. size: " << myints.size() << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **priority\_queue::top**

Пример использования метода top() в std::prioprity\_queue:

#include <iostream>

#include <queue>

int main() {

std::priority\_queue<int> mypq;

mypq.push(10);

mypq.push(20);

mypq.push(15);

std::cout << "mypq.top() is now " << mypq.top() << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **priority\_queue::push**

Пример использования методов push() и pop() в std::prioprity\_queue:

#include <iostream>

#include <queue>

int main() {

std::priority\_queue<int> mypq;

mypq.push(30);

mypq.push(100);

mypq.push(25);

mypq.push(40);

std::cout << "Popping out elements...";

while (!mypq.empty()) {

std::cout << ' ' << mypq.top();

mypq.pop();

}

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **priority\_queue::emplace**

Пример использования метода emplace() в std::prioprity\_queue:

#include <iostream>

#include <queue>

#include <string>

int main() {

std::priority\_queue<std::string> mypq;

mypq.emplace("orange");

mypq.emplace("strawberry");

mypq.emplace("apple");

mypq.emplace("pear");

std::cout << "mypq contains:";

while (!mypq.empty()) {

std::cout << ' ' << mypq.top();

mypq.pop();

}

std::cout << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

#### **priority\_queue::pop**

#### **priority\_queue::swap**

Пример использования метода swap() в std::prioprity\_queue:

#include <iostream>

#include <queue>

int main() {

std::priority\_queue<int> foo, bar;

foo.push(15); foo.push(30); foo.push(10);

bar.push(101); bar.push(202);

foo.swap(bar);

std::cout << "size of foo: " << foo.size() << '\n';

std::cout << "size of bar: " << bar.size() << '\n';

return 0;

}

Результат работы программы:



[<Адаптеры для контейнеров>](#_Адаптеры_для_контейнеров)

# *Последовательные контейнеры*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Members | | <array> | <vector> | <deque> | <forward\_list> | <list> |
| Алгоритмы, не изменяющие последовательность | adjacent\_find | [adjacent\_find](#_adjacent_find) | [adjacent\_find](#_adjacent_find_1) | [adjacent\_find](#_adjacent_find_2) |  | [adjacent\_find](#_adjacent_find_3) |
| count | [count](#_count) | [count](#_count_1) | [count](#_count_2) |  | [count](#_count_3) |
| count\_if | [count\_if](#_count_if) | [count\_if](#_count_if_1) | [count\_if](#_count_if_2) |  | [count\_if](#_count_if_3) |
| equal | [equal](#_equal) | [equal](#_equal_1) | [equal](#_equal_2) |  | [equal](#_equal_3) |
| find | [find](#_find) | [find](#_find_1) | [find](#_find_2) | [find](#_find_3) | [find](#_find_4) |
| find\_if | [find\_if](#_find_if) | [find\_if](#_find_if_1) | [find\_if](#_find_if_2) | [find\_if](#_find_if_3) | [find\_if](#_find_if_4) |
| find\_end | [find\_end](#_find_end) | [find\_end](#_find_end_1) | [find\_end](#_find_end_2) |  | [find\_end](#_find_end_3) |
| find\_first\_of | [find\_first\_of](#_find_first_of) | [find\_first\_of](#_find_first_of_1) | [find\_first\_of](#_find_first_of_2) |  | [find\_first\_of](#_find_first_of_3) |
| for\_each | [for\_each](#_for_each) | [for\_each](#_for_each_1) | [for\_each](#_for_each_2) | [for\_each](#_for_each_3) | [for\_each](#_for_each_4) |
| mismatch | [mismatch](#_mismatch) | [mismatch](#_mismatch_1) | [mismatch](#_mismatch_2) | [mismatch](#_mismatch_3) | [mismatch](#_mismatch_4) |
| search | [search](#_search) | [search](#_search_1) | [search](#_search_2) |  | [search](#_search_3) |
| search\_n | [search\_n](#_search_n) | [search\_n](#_search_n_1) | [search\_n](#_search_n_2) |  | [search\_n](#_search_n_3) |
| Минимум и максимум | max\_element | [max\_element](#_max_element) | [max\_element](#_max_element_1) | [max\_element](#_max_element_2) |  | [max\_element](#_max_element_3) |
| min\_element | [min\_element](#_min_element) | [min\_element](#_min_element_1) | [min\_element](#_min_element_2) |  | [min\_element](#_min_element_3) |
| Обобщенные численные алгоритмы | accumulate | [accumulate](#_accumulate) | [accumulate](#_accumulate_1) | [accumulate](#_accumulate_2) |  | [accumulate](#_accumulate_3) |
| adjacent\_difference | [adjacent\_difference](#_adjacent_difference) | [adjacent\_difference](#_adjacent_difference_1) | [adjacent\_difference](#_adjacent_difference_2) |  | [adjacent\_difference](#_adjacent_difference_3) |
| inner\_product | [inner\_product](#_inner_product) | [inner\_product](#_inner_product_1) | [inner\_product](#_inner_product_2) |  | [inner\_product](#_inner_product_3) |
| partial\_sum | [partial\_sum](#_partial_sum) | [partial\_sum](#_partial_sum_1) | [partial\_sum](#_partial_sum_2) |  | [partial\_sum](#_partial_sum_3) |
| Алгоритмы перестановки | swap | [swap](#_swap) | [swap](#_swap_1) | [swap](#_swap_2) |  |  |
| iter\_swap | [iter\_swap](#_iter_swap) | [iter\_swap](#_iter_swap_1) | [iter\_swap](#_iter_swap_2) |  |  |
| swap\_ranges | [swap\_ranges](#_swap_ranges) | [swap\_ranges](#_swap_ranges_1) | [swap\_ranges](#_swap_ranges_2) |  |  |
| Алгоритмы замены | replace | [replace](#_replace) | [replace](#_replace_1) | [replace](#_replace_2) |  | [replace](#_replace_3) |
| replace\_if | [replace\_if](#_replace_if) | [replace\_if](#_replace_if_1) | [replace\_if](#_replace_if_2) |  | [replace\_if](#_replace_if_3) |
| replace\_copy | [replace\_copy](#_replace_copy) | [replace\_copy](#_replace_copy_1) | [replace\_copy](#_replace_copy_2) |  | [replace\_copy](#_replace_copy_3) |
| replace\_copy\_if | [replace\_copy\_if](#_replace_copy_if) | [replace\_copy\_if](#_replace_copy_if_1) | [replace\_copy\_if](#_replace_copy_if_2) |  | [replace\_copy\_if](#_replace_copy_if_3) |
| Алгоритмы удаления | remove | [remove](#_remove) | [remove](#_remove_1) | [remove](#_remove_2) |  | [remove](#_remove_3) |
| remove\_if |  | [remove\_if](#_remove_if_1) | [remove\_if](#_remove_if_2) |  | [remove\_if](#_remove_if_3) |
| remove\_copy | [remove\_copy](#_remove_copy) | [remove\_copy](#_remove_copy_1) | [remove\_copy](#_remove_copy_2) |  | [remove\_copy](#_remove_copy_3) |
| remove\_copy\_if | [remove\_copy\_if](#_remove_copy_if) | [remove\_copy\_if](#_remove_copy_if_1) | [remove\_copy\_if](#_remove_copy_if_2) |  | [remove\_copy\_if](#_remove_copy_if_3) |
| Алгоритмы, изменяющие последовательность | copy | [copy](#_copy) | [copy](#_copy_1) | [copy](#_copy_2) | [copy](#_copy_4) | [copy](#_copy_3) |
| copy\_backward | [copy\_backward](#_copy_backward) | [copy\_backward](#_copy_backward_1) | [copy\_backward](#_copy_backward_2) |  |  |
| fill | [fill](#_fill) | [fill](#_fill_1) | [fill](#_fill_2) |  | [fill](#_fill_3) |
| fill\_n | [fill\_n](#_fill_n) | [fill\_n](#_fill_n_1) | [fill\_n](#_fill_n_2) |  | [fill\_n](#_fill_n_3) |
| generate | [generate](#_generate) | [generate](#_generate_1) | [generate](#_generate_2) | [generate](#_generate_3) | [generate](#_generate_4) |
| generate\_n | [generate\_n](#_generate_n) | [generate\_n](#_generate_n_1) | [generate\_n](#_generate_n_2) | [generate\_n](#_generate_n_3) | [generate\_n](#_generate_n_4) |
| partition | [partition](#_partition) | [partition](#_partition_1) | [partition](#_partition_2) |  | [partition](#_partition_3) |
| random\_shuffle | [random\_shuffle](#_random_shuffle) | [random\_shuffle](#_random_shuffle_1) | [random\_shuffle](#_random_shuffle_2) |  | [random\_shuffle](#_random_shuffle_3) |
| reverse | [reverse](#_reverse) | [reverse](#_reverse_1) | [reverse](#_reverse_2) | [reverse](#_reverse_3) | [reverse](#_reverse_4) |
| reverse\_copy | [reverse\_copy](#_reverse_copy) | [reverse\_copy](#_reverse_copy_1) | [reverse\_copy](#_reverse_copy_2) |  | [reverse\_copy](#_reverse_copy_3) |
| rotate | [rotate](#_rotate) | [rotate](#_rotate_1) | [rotate](#_rotate_2) |  | [rotate](#_rotate_3) |
| rotate\_copy | [rotate\_copy](#_rotate_copy) | [rotate\_copy](#_rotate_copy_1) | [rotate\_copy](#_rotate_copy_2) |  | [rotate\_copy](#_rotate_copy_3) |
| stable\_partition | [stable\_partition](#_stable_partition) | [stable\_partition](#_stable_partition_1) | [stable\_partition](#_stable_partition_2) | [stable\_partition](#_stable_partition_3) | [stable\_partition](#_stable_partition_4) |
| transform | [transform](#_transform) | [transform](#_transform_1) | [transform](#_transform_2) | [transform](#_transform_3) | [transform](#_transform_4) |
| unique | [unique](#_unique) | [unique](#_unique_1) | [unique](#_unique_2) | [unique](#_unique_3) | [unique](#_unique_4) |
| unique\_copy | [unique\_copy](#_unique_copy) | [unique\_copy](#_unique_copy_1) | [unique\_copy](#_unique_copy_2) | [unique\_copy](#_unique_copy_3) | [unique\_copy](#_unique_copy_4) |
| Алгоритмы сортировки | inplace\_merge | [inplace\_merge](#_inplace_merge) | [inplace\_merge](#_inplace_merge_1) | [inplace\_merge](#_inplace_merge_2) |  | [inplace\_merge](#_inplace_merge_3) |
| is\_sorted | [is\_sorted](#_is_sorted) | [is\_sorted](#_is_sorted_1) | [is\_sorted](#_is_sorted_2) |  | [is\_sorted](#_is_sorted_3) |
| lexicographical\_compare | [lexicographical\_compare](#_lexicographical_compare) | [lexicographical\_compare](#_lexicographical_compare_1) | [lexicographical\_compare](#_lexicographical_compare_2) |  | [lexicographical\_compare](#_lexicographical_compare_3) |
| merge |  | [merge](#_merge) |  |  | [merge](#_merge_1) |
| nth\_element | [nth\_element](#_nth_element) | [nth\_element](#_nth_element_1) | [nth\_element](#_nth_element_2) |  |  |
| partial\_sort | [partial\_sort](#_partial_sort) | [partial\_sort](#_partial_sort_1) | [partial\_sort](#_partial_sort_2) |  |  |
| partial\_sort\_copy | [partial\_sort\_copy](#_partial_sort_copy) | [partial\_sort\_copy](#_partial_sort_copy_1) | [partial\_sort\_copy](#_partial_sort_copy_2) |  |  |
| sort | [sort](#_sort) | [sort](#_sort_1) | [sort](#_sort_2) |  | [sort](#_sort_3) |
| stable\_sort | [stable\_sort](#_stable_sort) | [stable\_sort](#_stable_sort_1) | [stable\_sort](#_stable_sort_2) |  | [stable\_sort](#_stable_sort_3) |
| Алгоритмы для бинарного поиска | binary\_search | [binary\_search](#_binary_search) | [binary\_search](#_binary_search_1) | [binary\_search](#_binary_search_2) |  | [binary\_search](#_binary_search_3) |
| equal\_range | [equal\_range](#_equal_range) | [equal\_range](#_equal_range_1) | [equal\_range](#_equal_range_2) |  | [equal\_range](#_equal_range_3) |
| lower\_bound | [lower\_bound](#_lower_bound) | [lower\_bound](#_lower_bound_1) | [lower\_bound](#_lower_bound_2) |  | [lower\_bound](#_lower_bound_3) |
| upper\_bound | [upper\_bound](#_upper_bound) | [upper\_bound](#_upper_bound_1) | [upper\_bound](#_upper_bound_2) | [upper\_bound](#_upper_bound_3) | [upper\_bound](#_upper_bound_4) |

# *Ассоциативные контейнеры*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Members | | <set> | | <map> | |
| set | multiset | map | multimap |
| Алгоритмы, не изменяющие последовательность | adjacent\_find | [adjacent\_find](#_adjacent_find_4) | [adjacent\_find](#_adjacent_find_6) | [adjacent\_find](#_adjacent_find_5) |  |
| count | [count](#_count_4) | [count](#_count_5) |  |  |
| count\_if | [count\_if](#_count_if_4) | [count\_if](#_count_if_6) | [count\_if](#_count_if_5) |  |
| equal | [equal](#_equal_4) | [equal](#_equal_6) | [equal](#_equal_5) |  |
| find | [find](#_find_5) | [find](#_find_7) | [find](#_find_6) |  |
| find\_if | [find\_if](#_find_if_5) | [find\_if](#_find_if_7) | [find\_if](#_find_if_6) | [find\_if](#_find_if_9) |
| find\_end | [find\_end](#_find_end_4) |  | [find\_end](#_find_end_5) | [find\_end](#_find_end_7) |
| find\_first\_of | [find\_first\_of](#_find_first_of_4) | [find\_first\_of](#_find_first_of_6) | [find\_first\_of](#_find_first_of_5) |  |
| for\_each | [for\_each](#_for_each_5) | [for\_each](#_for_each_7) | [for\_each](#_for_each_6) | [for\_each](#_for_each_9) |
| mismatch | [mismatch](#_mismatch_5) | [mismatch](#_mismatch_7) | [mismatch](#_mismatch_6) | [mismatch](#_mismatch_9) |
| search | [search](#_search_4) |  | [search](#_search_5) | [search](#_search_7) |
| search\_n | [search\_n](#_search_n_4) |  | [search\_n](#_search_n_5) |  |
| Минимум и максимум | max\_element | [max\_element](#_max_element_4) | [max\_element](#_max_element_6) | [max\_element](#_max_element_5) | [max\_element](#_max_element_8) |
| min\_element | [min\_element](#_min_element_4) | [min\_element](#_min_element_6) | [min\_element](#_min_element_5) | [min\_element](#_min_element_8) |
| Обобщенные численные алгоритмы | accumulate | [accumulate](#_accumulate_4) | [accumulate](#_accumulate_6) | [accumulate](#_accumulate_5) | [accumulate](#_accumulate_8) |
| adjacent\_difference | [adjacent\_difference](#_adjacent_difference_4) | [adjacent\_difference](#_adjacent_difference_5) |  |  |
| inner\_product | [inner\_product](#_inner_product_4) | [inner\_product](#_inner_product_5) |  |  |
| partial\_sum | [partial\_sum](#_partial_sum_4) | [partial\_sum](#_partial_sum_5) |  |  |
| Алгоритмы перестановки | swap | [swap](#_swap_3) |  | [swap](#_swap_4) | [swap](#_swap_6) |
| iter\_swap |  |  |  |  |
| swap\_ranges |  |  |  |  |
| Алгоритмы замены | replace |  |  |  |  |
| replace\_if |  |  |  |  |
| replace\_copy | [replace\_copy](#_replace_copy_4) |  |  |  |
| replace\_copy\_if | [replace\_copy\_if](#_replace_copy_if_4) |  | [replace\_copy\_if](#_replace_copy_if_5) |  |
| Алгоритмы удаления | remove |  |  |  |  |
| remove\_if |  |  |  |  |
| remove\_copy | [remove\_copy](#_remove_copy_4) |  |  |  |
| remove\_copy\_if | [remove\_copy\_if](#_remove_copy_if_4) |  | [remove\_copy\_if](#_remove_copy_if_5) |  |
| Алгоритмы, изменяющие последовательность | copy | [copy](#_copy_5) | [copy](#_copy_7) | [copy](#_copy_6) | [copy](#_copy_9) |
| copy\_backward | [copy\_backward](#_copy_backward_3) |  |  |  |
| fill |  |  |  |  |
| fill\_n |  |  |  |  |
| generate |  |  |  |  |
| generate\_n |  |  |  |  |
| partition |  |  |  |  |
| random\_shuffle |  |  |  |  |
| reverse |  |  |  |  |
| reverse\_copy |  |  |  | [reverse\_copy](#_reverse_copy_5) |
| rotate |  |  |  |  |
| rotate\_copy |  |  |  |  |
| stable\_partition |  |  |  |  |
| transform | [transform](#_transform_5) | [transform](#_transform_7) | [transform](#_transform_6) | [transform](#_transform_9) |
| unique |  |  |  |  |
| unique\_copy | [unique\_copy](#_uniqie_copy) |  | [unique\_copy](#_unique_copy_5) | [unique\_copy](#_unique_copy_7) |
| Алгоритмы сортировки | inplace\_merge |  |  |  |  |
| is\_sorted | [is\_sorted](#_is_sorted_4) |  | [is\_sorted](#_is_sorted_5) | [is\_sorted](#_is_sorted_7) |
| lexicographical\_compare | [lexicographical\_compare](#_lexicographical_compare_4) | [lexicographical\_compare](#_lexicographical_compare_6) | [lexicographical\_compare](#_lexicographical_compare_5) | [lexicographical\_compare](#_lexicographical_compare_8) |
| merge | [merge](#_merge_2) | [merge](#_merge_4) | [merge](#_merge_3) | [merge](#_merge_6) |
| nth\_element |  |  |  |  |
| partial\_sort |  |  |  |  |
| partial\_sort\_copy | [partial\_sort\_copy](#_partial_sort_copy_3) |  | [partial\_sort\_copy](#_partial_sort_copy_4) |  |
| sort |  |  |  |  |
| stable\_sort |  |  |  |  |
| Алгоритмы для бинарного поиска | binary\_search | [binary\_search](#_binary_search_4) |  |  |  |
| equal\_range | [equal\_range](#_equal_range_4) |  | [equal\_range](#_equal_range_5) |  |
| lower\_bound | [lower\_bound](#_lower_bound_4) | [lower\_bound](#_lower_bound_6) | [lower\_bound](#_lower_bound_5) | [lower\_bound](#_lower_bound_8) |
| upper\_bound | [upper\_bound](#_upper_bound_5) | [upper\_bound](#_upper_bound_7) | [upper\_bound](#_upper_bound_6) |  |

# *Неупорядоченные ассоциативные контейнеры*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Members | | <unordered\_set> | | <unordered\_map> | |
| unordered\_set | unordered\_miltiset | unordered\_map | unordered\_miltimap |
| Алгоритмы, не изменяющие последовательность | adjacent\_find | [adjacent\_find](#_adjacent_find_7) | [adjacent\_find](#_adjacent_find_8) | [adjacent\_find](#_adjacent_find_9) |  |
| count | [count](#_count_6) | [count](#_count_7) | [count](#_count_8) |  |
| count\_if | [count\_if](#_count_if_7) | [count\_if](#_count_if_8) | [count\_if](#_count_if_9) | [count\_if](#_count_if_10) |
| equal | [equal](#_equal_7) | [equal](#_equal_8) | [equal](#_equal_9) | [equal](#_equal_10) |
| find | [find](#_find_8) | [find](#_find_9) | [find](#_find_10) |  |
| find\_if | [find\_if](#_find_if_8) | [find\_if](#_find_if_10) | [find\_if](#_find_if_11) | [find\_if](#_find_if_12) |
| find\_end | [find\_end](#_find_end_6) |  | [find\_end](#_find_end_8) | [find\_end](#_find_end_9) |
| find\_first\_of | [find\_first\_of](#_find_first_of_7) | [find\_first\_of](#_find_first_of_8) |  | [find\_first\_of](#_find_first_of_9) |
| for\_each | [for\_each](#_for_each_8) | [for\_each](#_for_each_10) | [for\_each](#_for_each_11) | [for\_each](#_for_each_12) |
| mismatch | [mismatch](#_mismatch_8) | [mismatch](#_mismatch_10) | [mismatch](#_mismatch_11) | [mismatch](#_mismatch_12) |
| search | [search](#_search_6) | [search](#_seacrh) | [search](#_search_8) | [search](#_search_9) |
| search\_n | [search\_n](#_search_n_6) | [search\_n](#_search_n_7) | [search\_n](#_search_n_8) | [search\_n](#_search_n_9) |
| Минимум и максимум | max\_element | [max\_element](#_max_element_7) | [max\_element](#_max_element_9) | [max\_element](#_max_element_10) | [max\_element](#_max_element_11) |
| min\_element | [min\_element](#_min_element_7) | [min\_element](#_min_element_9) | [min\_element](#_min_element_10) | [min\_element](#_min_element_11) |
| Обобщенные численные алгоритмы | accumulate | [accumulate](#_accumulate_7) | [accumulate](#_accumulate_9) | [accumulate](#_accumulate_10) | [accumulate](#_accumulate_11) |
| adjacent\_difference | [adjacent\_difference](#_adjacent_difference_6) | [adjacent\_difference](#_adjacent_difference_7) |  |  |
| inner\_product | [inner\_product](#_inner_product_6) | [inner\_product](#_inner_product_7) | [inner\_product](#_inner_product_8) |  |
| partial\_sum | [partial\_sum](#_partial_sum_6) | [partial\_sum](#_partial_sum_7) |  |  |
| Алгоритмы перестановки | swap | [swap](#_swap_5) | [swap](#_swap_7) | [swap](#_swap_8) | [swap](#_swap_9) |
| iter\_swap |  |  |  |  |
| swap\_ranges |  |  |  |  |
| Алгоритмы замены | replace |  |  |  |  |
| replace\_if |  |  |  |  |
| replace\_copy | [replace\_copy](#_replace_copy_5) | [replace\_copy](#_replacy_copy) |  |  |
| replace\_copy\_if | [replace\_copy\_if](#_replace_copy_if_6) | [replace\_copy\_if](#_replace_copy_if_7) |  |  |
| Алгоритмы удаления | remove |  | [remove](#_remove_4) |  |  |
| remove\_if |  |  |  |  |
| remove\_copy | [remove\_copy](#_remove_copy_5) | [remove\_copy](#_remove_copy_6) |  |  |
| remove\_copy\_if | [remove\_copy\_if](#_remove_copy_if_6) | [remove\_copy\_if](#_remove_copy_if_7) | [remove\_copy\_if](#_remove_copy_if_8) |  |
| Алгоритмы, изменяющие последовательность | copy | [copy](#_copy_8) | [copy](#_copy_10) | [copy](#_copy_11) | [copy](#_copy_12) |
| copy\_backward | [copy\_backward](#_copy_backward_4) |  |  |  |
| fill |  |  |  |  |
| fill\_n |  |  | [fill\_n](#_fill_n_4) |  |
| generate |  |  |  |  |
| generate\_n |  |  | [generate\_n](#_genetate_n) |  |
| partition |  |  |  |  |
| random\_shuffle |  |  |  |  |
| reverse |  |  |  |  |
| reverse\_copy | [reverse\_copy](#_reverse_copy_4) | [reverse\_copy](#_reverse_copy_6) | [reverse\_copy](#_reverse_copy_7) |  |
| rotate |  |  |  |  |
| rotate\_copy |  |  |  |  |
| stable\_partition |  |  |  |  |
| transform | [transform](#_transform_8) | [transform](#_transform_10) |  | [transform](#_transform_11) |
| unique |  |  |  |  |
| unique\_copy | [unique\_copy](#_unique_copy_6) |  |  |  |
| Алгоритмы сортировки | inplace\_merge |  |  |  |  |
| is\_sorted | [is\_sorted](#_is_sorted_6) | [is\_sorted](#_is_sorted_8) | [is\_sorted](#_is_sorted_9) |  |
| lexicographical\_compare | [lexicographical\_compare](#_lexicographical_compare_7) | [lexicographical\_compare](#_lexicographical_compare_9) | [lexicographical\_compare](#_lexicographical_compare_10) | [lexicographical\_compare](#_lexicographical_compare_11) |
| merge | [merge](#_merge_5) |  |  | [merge](#_merge_7) |
| nth\_element |  |  |  |  |
| partial\_sort |  |  |  |  |
| partial\_sort\_copy | [partial\_sort\_copy](#_partial_sort_copy_5) |  |  |  |
| sort |  |  |  |  |
| stable\_sort |  |  |  |  |
| Алгоритмы для бинарного поиска | binary\_search |  | [binary\_search](#_binary_search_5) |  |  |
| equal\_range | [equal\_range](#_equal_range_6) | [equal\_range](#_equal_range_7) | [equal\_range](#_equal_range_8) | [equal\_range](#_equal_range_9) |
| lower\_bound | [lower\_bound](#_lower_bound_7) |  | [lower\_bound](#_lower_bound_9) |  |
| upper\_bound | [upper\_bound](#_upper_bound_8) |  |  |  |

# **Последовательные контейнеры**

## **Array**

#### **adjacent\_find**

Пример использования алгоритма adjacent\_find:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 6> arr = { 1, 2, 3, 3, 4, 5 };

auto result = std::adjacent\_find(arr.begin(), arr.end());

if (result != arr.end()) {

std::cout << "Найдена пара соседних элементов, равных друг другу: "

<< \*result << " и " << \*(result + 1) << std::endl;

}

else {

std::cout << "Соседние элементы, равные друг другу, не найдены." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **count**

Пример использования алгоритма count:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 6> arr = { 1, 2, 3, 3, 4, 5 };

int value = 3;

int count = std::count(arr.begin(), arr.end(), value);

std::cout << "Количество вхождений элемента " << value << " в массиве: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **count\_if**

Пример использования алгоритма coun\_if:

#include <iostream>

#include <array>

#include <algorithm>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 6> arr = { 1, 2, 3, 4, 5, 6 };

int count = std::count\_if(arr.begin(), arr.end(), isEven);

std::cout << "Количество четных элементов в массиве: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **equal**

Пример использования алгоритма equal:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr1 = { 1, 2, 3, 4, 5 };

std::array<int, 5> arr2 = { 1, 2, 3, 4, 5 };

bool isEqual = std::equal(arr1.begin(), arr1.end(), arr2.begin());

std::cout << "Массивы arr1 и arr2 " << (isEqual ? "равны" : "не равны") << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find**

Пример использования алгоритма find:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr = { 1, 2, 3, 4, 5 };

auto it = std::find(arr.begin(), arr.end(), 3);

if (it != arr.end()) {

std::cout << "Элемент 3 найден в массиве arr." << std::endl;

}

else {

std::cout << "Элемент 3 не найден в массиве arr." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_if**

Пример использования алгоритма find\_if:

#include <iostream>

#include <array>

#include <algorithm>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr = { 1, 2, 3, 4, 5 };

auto it = std::find\_if(arr.begin(), arr.end(), isEven);

if (it != arr.end()) {

std::cout << "Первое четное число в массиве arr: " << \*it << std::endl;

}

else {

std::cout << "Четное число не найдено в массиве arr." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_end**

Пример использования алгоритма find\_end:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 10> arr = { 1, 2, 3, 4, 5, 3, 4, 5, 3, 4 };

std::array<int, 2> subArr = { 3, 4 };

auto it = std::find\_end(arr.begin(), arr.end(), subArr.begin(), subArr.end());

if (it != arr.end()) {

std::cout << "Подмассив найден в массиве arr, начиная с индекса: " << std::distance(arr.begin(), it) << std::endl;

}

else {

std::cout << "Подмассив не найден в массиве arr." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_first\_of**

Пример использования алгоритма find\_first\_of:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 10> arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

std::array<int, 3> searchValues = { 4, 8, 12 };

auto it = std::find\_first\_of(arr.begin(), arr.end(), searchValues.begin(), searchValues.end());

if (it != arr.end()) {

std::cout << "Первое вхождение одного из искомых значений в массиве arr: " << \*it << std::endl;

}

else {

std::cout << "Искомые значения не найдены в массиве arr." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **for\_each**

Пример использования алгоритма for\_each:

#include <iostream>

#include <array>

#include <algorithm>

void printSquare(int num) {

std::cout << num \* num << " ";

}

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr = { 1, 2, 3, 4, 5 };

std::cout << "Исходный массив: ";

for (int num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Квадраты элементов массива: ";

std::for\_each(arr.begin(), arr.end(), printSquare);

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **mismatch**

Пример использования алгоритма mismatch:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr1 = { 1, 2, 3, 4, 5 };

std::array<int, 5> arr2 = { 1, 2, 3, 6, 5 };

auto pair = std::mismatch(arr1.begin(), arr1.end(), arr2.begin());

if (pair.first == arr1.end()) {

std::cout << "Все элементы массивов совпадают." << std::endl;

}

else {

std::cout << "Первое несовпадение: " << \*pair.first << " и " << \*pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **search**

Пример использования алгоритма search:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 8> arr = { 1, 2, 3, 4, 5, 6, 7, 8 };

std::array<int, 3> pattern = { 4, 5, 6 };

auto result = std::search(arr.begin(), arr.end(), pattern.begin(), pattern.end());

if (result != arr.end()) {

std::cout << "Найдено совпадение начиная с позиции " << std::distance(arr.begin(), result) << std::endl;

}

else {

std::cout << "Совпадение не найдено." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **search\_n**

Пример использования алгоритма search\_n:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 8> arr = { 1, 2, 3, 4, 4, 4, 5, 6 };

int value = 4;

int count = 3;

auto result = std::search\_n(arr.begin(), arr.end(), count, value);

if (result != arr.end()) {

std::cout << "Найдено совпадение " << count << " элементов со значением " << value << " начиная с позиции " << std::distance(arr.begin(), result) << std::endl;

}

else {

std::cout << "Совпадение не найдено." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **max\_element**

Пример использования алгоритма max\_element:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr = { 3, 1, 4, 1, 5 };

int maxValue = \*std::max\_element(arr.begin(), arr.end());

std::cout << "Максимальное значение: " << maxValue << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **min\_element**

Пример использования алгоритма min\_element:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr = { 3, 1, 4, 1, 5 };

int minValue = \*std::min\_element(arr.begin(), arr.end());

std::cout << "Минимальное значение: " << minValue << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **accumulate**

Пример использования алгоритма accumulate:

#include <iostream>

#include <array>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr = { 1, 2, 3, 4, 5 };

int sum = std::accumulate(arr.begin(), arr.end(), 0);

std::cout << "Сумма элементов: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **adjacent\_difference**

Пример использования алгоритма adjacent\_difference:

#include <iostream>

#include <array>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr = { 1, 3, 5, 7, 9 };

std::array<int, 5> diff;

std::adjacent\_difference(arr.begin(), arr.end(), diff.begin());

std::cout << "Разности между соседними элементами: ";

for (const auto& num : diff) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **inner\_product**

Пример использования алгоритма inner\_product:

#include <iostream>

#include <array>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr1 = { 1, 2, 3, 4, 5 };

std::array<int, 5> arr2 = { 6, 7, 8, 9, 10 };

int result = std::inner\_product(arr1.begin(), arr1.end(), arr2.begin(), 0);

std::cout << "Внутреннее произведение: " << result << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partial\_sum**

Пример использования алгоритма partial\_sum:

#include <iostream>

#include <array>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr = { 1, 2, 3, 4, 5 };

std::array<int, 5> result;

std::partial\_sum(arr.begin(), arr.end(), result.begin());

std::cout << "Частичные суммы: ";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **swap**

Пример использования алгоритма swap:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr1 = { 1, 2, 3, 4, 5 };

std::array<int, 5> arr2 = { 6, 7, 8, 9, 10 };

std::cout << "До обмена:\n";

std::cout << "arr1: ";

for (const auto& num : arr1) {

std::cout << num << " ";

}

std::cout << "\narr2: ";

for (const auto& num : arr2) {

std::cout << num << " ";

}

std::cout << std::endl;

std::swap(arr1, arr2);

std::cout << "После обмена:\n";

std::cout << "arr1: ";

for (const auto& num : arr1) {

std::cout << num << " ";

}

std::cout << "\narr2: ";

for (const auto& num : arr2) {

std::cout << num << " ";

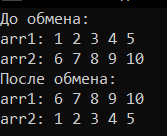
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **iter\_swap**

Пример использования алгоритма iter\_swap:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr = { 1, 2, 3, 4, 5 };

auto it1 = arr.begin();

auto it2 = arr.begin() + 3;

std::cout << "До обмена:\n";

std::cout << "arr: ";

for (const auto& num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

std::iter\_swap(it1, it2);

std::cout << "После обмена:\n";

std::cout << "arr: ";

for (const auto& num : arr) {

std::cout << num << " ";

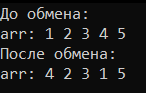
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **swap\_ranges**

Пример использования алгоритма swap\_ranges:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr1 = { 1, 2, 3, 4, 5 };

std::array<int, 5> arr2 = { 6, 7, 8, 9, 10 };

std::cout << "До обмена:\n";

std::cout << "arr1: ";

for (const auto& num : arr1) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "arr2: ";

for (const auto& num : arr2) {

std::cout << num << " ";

}

std::cout << std::endl;

std::swap\_ranges(arr1.begin(), arr1.end(), arr2.begin());

std::cout << "После обмена:\n";

std::cout << "arr1: ";

for (const auto& num : arr1) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "arr2: ";

for (const auto& num : arr2) {

std::cout << num << " ";

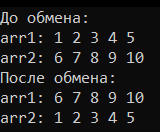
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace**

Пример использования алгоритма replace:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 6> arr = { 1, 2, 3, 4, 5, 3 };

int oldValue = 3;

int newValue = 9;

std::cout << "До замены:\n";

for (const auto& num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

std::replace(arr.begin(), arr.end(), oldValue, newValue);

std::cout << "После замены:\n";

for (const auto& num : arr) {

std::cout << num << " ";

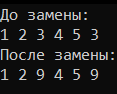
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_if**

Пример использования алгоритма replace\_if:

#include <iostream>

#include <array>

#include <algorithm>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 6> arr = { 1, 2, 3, 4, 5, 6 };

int newValue = 0;

std::cout << "До замены:\n";

for (const auto& num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

std::replace\_if(arr.begin(), arr.end(), isEven, newValue);

std::cout << "После замены:\n";

for (const auto& num : arr) {

std::cout << num << " ";

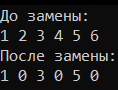
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_copy**

Пример использования алгоритма replace\_copy:

#include <iostream>

#include <array>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 6> arr = { 1, 2, 3, 4, 5, 6 };

std::array<int, 6> newArr;

int oldValue = 2;

int newValue = 0;

std::replace\_copy(arr.begin(), arr.end(), newArr.begin(), oldValue, newValue);

std::cout << "Исходный массив:\n";

for (const auto& num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Новый массив после замены:\n";

for (const auto& num : newArr) {

std::cout << num << " ";

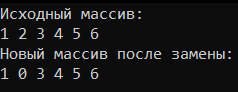
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_copy\_if**

Пример использования алгоритма replace\_copy\_if:

#include <iostream>

#include <array>

#include <algorithm>

#include <iterator>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 6> arr = { 1, 2, 3, 4, 5, 6 };

std::array<int, 6> newArr;

int newValue = 0;

std::replace\_copy\_if(arr.begin(), arr.end(), newArr.begin(), isEven, newValue);

std::cout << "Исходный массив:\n";

for (const auto& num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Новый массив после замены:\n";

for (const auto& num : newArr) {

std::cout << num << " ";

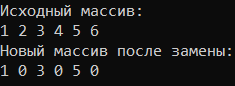
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove**

Пример использования алгоритма remove:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 7> arr = { 1, 2, 3, 2, 4, 2, 5 };

int valueToRemove = 2;

auto newEnd = std::remove(arr.begin(), arr.end(), valueToRemove);

std::cout << "Исходный массив:\n";

for (const auto& num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Массив после удаления элементов:\n";

for (auto it = arr.begin(); it != newEnd; ++it) {

std::cout << \*it << " ";

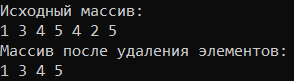
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove\_copy**

Пример использования алгоритма remove\_copy:

#include <iostream>

#include <array>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 7> arr = { 1, 2, 3, 4, 5, 6, 7 };

std::array<int, 7> result;

// Копируем все числа, кроме 3, в новый массив

std::remove\_copy(arr.begin(), arr.end(), result.begin(), 3);

std::cout << "Результирующий массив:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove\_copy\_if**

Пример использования алгоритма remove\_copy\_if:

#include <iostream>

#include <array>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 7> arr = { 1, 2, 3, 4, 5, 6, 7 };

std::array<int, 7> result;

// Копируем все нечетные числа в новый массив

std::remove\_copy\_if(arr.begin(), arr.end(), result.begin(), [](int num) { return num % 2 != 0; });

std::cout << "Результирующий массив:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **copy**

Пример использования алгоритма copy:

#include <iostream>

#include <array>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr = { 1, 2, 3, 4, 5 };

std::array<int, 5> result;

// Копируем элементы из исходного массива в новый массив

std::copy(arr.begin(), arr.end(), result.begin());

std::cout << "Результирующий массив:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **copy\_backward**

Пример использования алгоритма copy\_backward:

#include <iostream>

#include <array>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr = { 1, 2, 3, 4, 5 };

std::array<int, 5> result;

// Копируем элементы из исходного массива в новый массив в обратном порядке

std::copy\_backward(arr.begin(), arr.end(), result.end());

std::cout << "Результирующий массив:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **fill**

Пример использования алгоритма fill:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr;

// Заполняем весь массив значением 42

std::fill(arr.begin(), arr.end(), 42);

std::cout << "Массив после заполнения:\n";

for (const auto& num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **fill\_n**

Пример использования алгоритма fill\_n:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr;

// Заполняем первые 3 элемента массива значением 42

std::fill\_n(arr.begin(), 3, 42);

std::cout << "Массив после заполнения:\n";

for (const auto& num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **generate**

Пример использования алгоритма generate:

#include <iostream>

#include <array>

#include <algorithm>

int generateValue() {

static int value = 0;

return value++;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::array<int, 5> arr;

// Генерируем значения для каждого элемента массива с помощью функции generateValue

std::generate(arr.begin(), arr.end(), generateValue);

std::cout << "Массив после генерации значений:\n";

for (const auto& num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **generate\_n**

Пример использования алгоритма generate\_n:

#include <iostream>

#include <array>

#include <algorithm>

int generateValue() {

static int value = 0;

return value++;

}

int main() {

std::array<int, 5> arr;

std::generate\_n(arr.begin(), 5, generateValue);

std::cout << "Generated array: ";

for (const auto& element : arr) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partition**

Пример использования алгоритма partition:

#include <iostream>

#include <array>

#include <algorithm>

bool isEven(int number) {

return number % 2 == 0;

}

int main() {

std::array<int, 8> arr{ 1, 2, 3, 4, 5, 6, 7, 8 };

std::partition(arr.begin(), arr.end(), isEven);

std::cout << "Partitioned array: ";

for (const auto& element : arr) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **random\_shuffle**

Пример использования алгоритма random\_shuffle:

#include <iostream>

#include <array>

#include <algorithm>

#include <random>

int main() {

std::array<int, 5> arr{ 1, 2, 3, 4, 5 };

std::random\_device rd;

std::mt19937 gen(rd());

std::shuffle(arr.begin(), arr.end(), gen);

std::cout << "Shuffled array: ";

for (const auto& element : arr) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **reverse**

Пример использования алгоритма reverse:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 5> arr{ 1, 2, 3, 4, 5 };

std::reverse(arr.begin(), arr.end());

std::cout << "Reversed array: ";

for (const auto& element : arr) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **reverse\_copy**

Пример использования алгоритма reverse­\_copy:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 5> arr{ 1, 2, 3, 4, 5 };

std::array<int, 5> reversedArr;

std::reverse\_copy(arr.begin(), arr.end(), reversedArr.begin());

std::cout << "Reversed copy of array: ";

for (const auto& element : reversedArr) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **rotate**

Пример использования алгоритма rotate:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 5> arr{ 1, 2, 3, 4, 5 };

std::rotate(arr.begin(), arr.begin() + 2, arr.end());

std::cout << "Rotated array: ";

for (const auto& element : arr) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **rotate\_copy**

Пример использования алгоритма rotate\_copy:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 5> arr{ 1, 2, 3, 4, 5 };

std::array<int, 5> rotatedArr;

std::rotate\_copy(arr.begin(), arr.begin() + 2, arr.end(), rotatedArr.begin());

std::cout << "Original array: ";

for (const auto& element : arr) {

std::cout << element << " ";

}

std::cout << std::endl;

std::cout << "Rotated array: ";

for (const auto& element : rotatedArr) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **stable\_partition**

Пример использования алгоритма stable\_partition:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 5> arr{ 1, 2, 3, 4, 5 };

auto isEven = [](int num) { return num % 2 == 0; };

std::stable\_partition(arr.begin(), arr.end(), isEven);

std::cout << "Partitioned array: ";

for (const auto& element : arr) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 5> arr{ 1, 2, 3, 4, 5 };

std::array<int, 5> result;

std::transform(arr.begin(), arr.end(), result.begin(), [](int num) {

return num \* 2;

});

std::cout << "Transformed array: ";

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **unique**

Пример использования алгоритма unique:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 8> arr{ 1, 2, 2, 3, 3, 3, 4, 5 };

auto last = std::unique(arr.begin(), arr.end());

std::cout << "Unique elements in array: ";

for (auto it = arr.begin(); it != last; ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **unique\_copy**

Пример использования алгоритма unique\_copy:

#include <iostream>

#include <array>

#include <algorithm>

#include <iterator>

int main() {

std::array<int, 8> arr{ 1, 2, 2, 3, 3, 3, 4, 5 };

std::array<int, 8> result;

auto last = std::unique\_copy(arr.begin(), arr.end(), result.begin());

std::cout << "Unique elements in array: ";

for (auto it = result.begin(); it != last; ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **inplace\_merge**

Пример использования алгоритма inplace\_merge:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 8> arr1{ 1, 3, 5, 7 };

std::array<int, 4> arr2{ 2, 4, 6, 8 };

std::cout << "Before merge: ";

for (const auto& element : arr1) {

std::cout << element << " ";

}

for (const auto& element : arr2) {

std::cout << element << " ";

}

std::cout << std::endl;

std::inplace\_merge(arr1.begin(), arr1.end(), arr1.end() + arr2.size());

std::cout << "After merge: ";

for (const auto& element : arr1) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **is\_sorted**

Пример использования алгоритма is\_sorted:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 5> arr{ 1, 2, 3, 4, 5 };

if (std::is\_sorted(arr.begin(), arr.end())) {

std::cout << "The array is sorted." << std::endl;

}

else {

std::cout << "The array is not sorted." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **lexicographical\_compare**

Пример использования алгоритма lexicographical\_copmare:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 5> arr1{ 1, 2, 3, 4, 5 };

std::array<int, 5> arr2{ 1, 2, 3, 5, 4 };

if (std::lexicographical\_compare(arr1.begin(), arr1.end(), arr2.begin(), arr2.end())) {

std::cout << "arr1 is lexicographically less than arr2." << std::endl;

}

else {

std::cout << "arr1 is lexicographically greater or equal to arr2." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **nth\_element**

Пример использования алгоритма nth\_element:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 6> arr{ 9, 7, 5, 3, 1, 8 };

std::nth\_element(arr.begin(), arr.begin() + 3, arr.end());

std::cout << "Partial sorted array: ";

for (int num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partial\_sort**

Пример использования алгоритма partial\_sort:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 6> arr{ 9, 7, 5, 3, 1, 8 };

std::partial\_sort(arr.begin(), arr.begin() + 3, arr.end());

std::cout << "Partial sorted array: ";

for (int num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partial\_sort\_copy**

Пример использования алгоритма partial\_sort\_copy:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 6> arr{ 9, 7, 5, 3, 1, 8 };

std::array<int, 3> result{};

std::partial\_sort\_copy(arr.begin(), arr.end(), result.begin(), result.end());

std::cout << "Partial sorted copy: ";

for (int num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **sort**

Пример использования алгоритма sort:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 6> arr{ 9, 7, 5, 3, 1, 8 };

std::sort(arr.begin(), arr.end());

std::cout << "Sorted array: ";

for (int num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **stable\_sort**

Пример использования алгоритма stable\_sort:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 6> arr{ 9, 7, 5, 3, 1, 8 };

std::stable\_sort(arr.begin(), arr.end());

std::cout << "Sorted array: ";

for (int num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **binary\_search**

Пример использования алгоритма binary\_search:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 6> arr{ 1, 3, 5, 7, 9, 11 };

int target = 7;

bool found = std::binary\_search(arr.begin(), arr.end(), target);

if (found) {

std::cout << "Value " << target << " found in the array." << std::endl;

}

else {

std::cout << "Value " << target << " not found in the array." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **equal\_range**

Пример использования алгоритма equal\_range:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 7> arr{ 1, 2, 3, 4, 4, 5, 6 };

auto range = std::equal\_range(arr.begin(), arr.end(), 4);

std::cout << "Range of elements equal to 4 in the array: [";

for (auto it = range.first; it != range.second; ++it) {

std::cout << \*it << " ";

}

std::cout << "]" << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **lower\_bound**

Пример использования алгоритма lower\_bound:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 7> arr{ 1, 2, 3, 4, 4, 5, 6 };

auto it = std::lower\_bound(arr.begin(), arr.end(), 4);

if (it != arr.end()) {

std::cout << "First element not less than 4 in the array: " << \*it << std::endl;

}

else {

std::cout << "No element found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **upper\_bound**

Пример использования алгоритма upper\_bound:

#include <iostream>

#include <array>

#include <algorithm>

int main() {

std::array<int, 7> arr{ 1, 2, 3, 4, 4, 5, 6 };

auto it = std::upper\_bound(arr.begin(), arr.end(), 4);

if (it != arr.end()) {

std::cout << "First element greater than 4 in the array: " << \*it << std::endl;

}

else {

std::cout << "No element found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

## **Vector**

#### **adjacent\_find**

Пример использования алгоритма adjacent\_find:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 3, 4, 5 };

auto result = std::adjacent\_find(vec.begin(), vec.end());

if (result != vec.end()) {

std::cout << "Найдена пара соседних элементов, равных друг другу: "

<< \*result << " и " << \*(result + 1) << std::endl;

}

else {

std::cout << "Соседние элементы, равные друг другу, не найдены." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **count**

Пример использования алгоритма count:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 3, 4, 5 };

int value = 3;

int count = std::count(vec.begin(), vec.end(), value);

std::cout << "Количество вхождений элемента " << value << " в векторе: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **count\_if**

Пример использования алгоритма count\_if:

#include <iostream>

#include <vector>

#include <algorithm>

bool isPositive(int num) {

return num > 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { -1, 2, -3, 4, -5, 6 };

int count = std::count\_if(vec.begin(), vec.end(), isPositive);

std::cout << "Количество положительных элементов в векторе: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **equal**

Пример использования алгоритма equal:

# #include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec1 = { 1, 2, 3, 4, 5 };

std::vector<int> vec2 = { 1, 2, 3, 4, 5 };

bool isEqual = std::equal(vec1.begin(), vec1.end(), vec2.begin());

std::cout << "Векторы vec1 и vec2 " << (isEqual ? "равны" : "не равны") << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find**

Пример использования алгоритма find:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5 };

auto it = std::find(vec.begin(), vec.end(), 3);

if (it != vec.end()) {

std::cout << "Элемент 3 найден в векторе vec." << std::endl;

}

else {

std::cout << "Элемент 3 не найден в векторе vec." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_if**

Пример использования алгоритма find\_if:

#include <iostream>

#include <vector>

#include <algorithm>

bool isPositive(int num) {

return num > 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { -2, -1, 0, 1, 2 };

auto it = std::find\_if(vec.begin(), vec.end(), isPositive);

if (it != vec.end()) {

std::cout << "Первое положительное число в векторе vec: " << \*it << std::endl;

}

else {

std::cout << "Положительное число не найдено в векторе vec." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_end**

Пример использования алгоритма find\_end:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5, 3, 4, 5, 3, 4 };

std::vector<int> subVec = { 3, 4 };

auto it = std::find\_end(vec.begin(), vec.end(), subVec.begin(), subVec.end());

if (it != vec.end()) {

std::cout << "Подвектор найден в векторе vec, начиная с индекса: " << std::distance(vec.begin(), it) << std::endl;

}

else {

std::cout << "Подвектор не найден в векторе vec." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_first\_of**

Пример использования алгоритма find\_first\_of:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

std::vector<int> searchValues = { 4, 8, 12 };

auto it = std::find\_first\_of(vec.begin(), vec.end(), searchValues.begin(), searchValues.end());

if (it != vec.end()) {

std::cout << "Первое вхождение одного из искомых значений в векторе vec: " << \*it << std::endl;

}

else {

std::cout << "Искомые значения не найдены в векторе vec." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **for\_each**

Пример использования алгоритма for\_each:

#include <iostream>

#include <vector>

#include <algorithm>

void printCube(int num) {

std::cout << num \* num \* num << " ";

}

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5 };

std::cout << "Исходный вектор: ";

for (int num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Кубы элементов вектора: ";

std::for\_each(vec.begin(), vec.end(), printCube);

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **mismatch**

Пример использования алгоритма mismatch:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec1 = { 1, 2, 3, 4, 5 };

std::vector<int> vec2 = { 1, 2, 3, 6, 5 };

auto pair = std::mismatch(vec1.begin(), vec1.end(), vec2.begin());

if (pair.first == vec1.end()) {

std::cout << "Все элементы векторов совпадают." << std::endl;

}

else {

std::cout << "Первое несовпадение: " << \*pair.first << " и " << \*pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **search**

Пример использования алгоритма search:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5, 6, 7, 8 };

std::vector<int> pattern = { 4, 5, 6 };

auto result = std::search(vec.begin(), vec.end(), pattern.begin(), pattern.end());

if (result != vec.end()) {

std::cout << "Найдено совпадение начиная с позиции " << std::distance(vec.begin(), result) << std::endl;

}

else {

std::cout << "Совпадение не найдено." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **search\_n**

Пример использования алгоритма search\_n:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 4, 4, 5, 6 };

int value = 4;

int count = 3;

auto result = std::search\_n(vec.begin(), vec.end(), count, value);

if (result != vec.end()) {

std::cout << "Найдено совпадение " << count << " элементов со значением " << value << " начиная с позиции " << std::distance(vec.begin(), result) << std::endl;

}

else {

std::cout << "Совпадение не найдено." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **max\_element**

Пример использования алгоритма max\_element:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 3, 1, 4, 1, 5 };

int maxValue = \*std::max\_element(vec.begin(), vec.end());

std::cout << "Максимальное значение: " << maxValue << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **min\_element**

Пример использования алгоритма min\_element:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 3, 1, 4, 1, 5 };

int minValue = \*std::min\_element(vec.begin(), vec.end());

std::cout << "Минимальное значение: " << minValue << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **accumulate**

Пример использования алгоритма accumulate:

#include <iostream>

#include <vector>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5 };

int sum = std::accumulate(vec.begin(), vec.end(), 0);

std::cout << "Сумма элементов: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **adjacent\_difference**

Пример использования алгоритма adjacent\_difference:

#include <iostream>

#include <vector>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 3, 5, 7, 9 };

std::vector<int> diff(vec.size());

std::adjacent\_difference(vec.begin(), vec.end(), diff.begin());

std::cout << "Разности между соседними элементами: ";

for (const auto& num : diff) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **inner\_product**

Пример использования алгоритма inner\_product:

#include <iostream>

#include <vector>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec1 = { 1, 2, 3, 4, 5 };

std::vector<int> vec2 = { 6, 7, 8, 9, 10 };

int result = std::inner\_product(vec1.begin(), vec1.end(), vec2.begin(), 0);

std::cout << "Внутреннее произведение: " << result << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partial\_sum**

Пример использования алгоритма partial\_sum:

#include <iostream>

#include <vector>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5 };

std::vector<int> result(vec.size());

std::partial\_sum(vec.begin(), vec.end(), result.begin());

std::cout << "Частичные суммы: ";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **swap**

Пример использования алгоритма swap:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec1 = { 1, 2, 3, 4, 5 };

std::vector<int> vec2 = { 6, 7, 8, 9, 10 };

std::cout << "До обмена:\n";

std::cout << "vec1: ";

for (const auto& num : vec1) {

std::cout << num << " ";

}

std::cout << "\nvec2: ";

for (const auto& num : vec2) {

std::cout << num << " ";

}

std::cout << std::endl;

std::swap(vec1, vec2);

std::cout << "После обмена:\n";

std::cout << "vec1: ";

for (const auto& num : vec1) {

std::cout << num << " ";

}

std::cout << "\nvec2: ";

for (const auto& num : vec2) {

std::cout << num << " ";

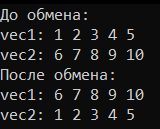
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **iter\_swap**

Пример использования алгоритма iter\_swap:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5 };

auto it1 = vec.begin();

auto it2 = vec.begin() + 3;

std::cout << "До обмена:\n";

std::cout << "vec: ";

for (const auto& num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

std::iter\_swap(it1, it2);

std::cout << "После обмена:\n";

std::cout << "vec: ";

for (const auto& num : vec) {

std::cout << num << " ";

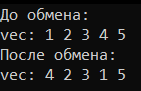
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **swap\_ranges**

Пример использования алгоритма swap\_ranges:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec1 = { 1, 2, 3, 4, 5 };

std::vector<int> vec2 = { 6, 7, 8, 9, 10 };

std::cout << "До обмена:\n";

std::cout << "vec1: ";

for (const auto& num : vec1) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "vec2: ";

for (const auto& num : vec2) {

std::cout << num << " ";

}

std::cout << std::endl;

std::swap\_ranges(vec1.begin(), vec1.end(), vec2.begin());

std::cout << "После обмена:\n";

std::cout << "vec1: ";

for (const auto& num : vec1) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "vec2: ";

for (const auto& num : vec2) {

std::cout << num << " ";

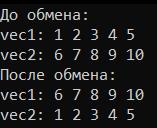
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace**

Пример использования алгоритма replace:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5, 3 };

int oldValue = 3;

int newValue = 9;

std::cout << "До замены:\n";

for (const auto& num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

std::replace(vec.begin(), vec.end(), oldValue, newValue);

std::cout << "После замены:\n";

for (const auto& num : vec) {

std::cout << num << " ";

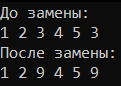
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_if**

Пример использования алгоритма replace\_if:

#include <iostream>

#include <vector>

#include <algorithm>

bool isNegative(int num) {

return num < 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { -1, 2, -3, 4, -5, 6 };

int newValue = 0;

std::cout << "До замены:\n";

for (const auto& num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

std::replace\_if(vec.begin(), vec.end(), isNegative, newValue);

std::cout << "После замены:\n";

for (const auto& num : vec) {

std::cout << num << " ";

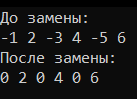
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_copy**

Пример использования алгоритма replace\_copy:

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5, 6 };

std::vector<int> newVec;

int oldValue = 2;

int newValue = 0;

std::replace\_copy(vec.begin(), vec.end(), std::back\_inserter(newVec), oldValue, newValue);

std::cout << "Исходный вектор:\n";

for (const auto& num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Новый вектор после замены:\n";

for (const auto& num : newVec) {

std::cout << num << " ";

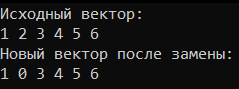
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_copy\_if**

Пример использования алгоритма replace\_copy\_if:

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5, 6 };

std::vector<int> newVec;

int newValue = 0;

std::replace\_copy\_if(vec.begin(), vec.end(), std::back\_inserter(newVec), isEven, newValue);

std::cout << "Исходный вектор:\n";

for (const auto& num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Новый вектор после замены:\n";

for (const auto& num : newVec) {

std::cout << num << " ";

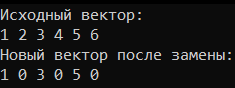
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove**

Пример использования алгоритма remove:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 2, 4, 2, 5 };

int valueToRemove = 2;

auto newEnd = std::remove(vec.begin(), vec.end(), valueToRemove);

std::cout << "Исходный вектор:\n";

for (const auto& num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Вектор после удаления элементов:\n";

for (auto it = vec.begin(); it != newEnd; ++it) {

std::cout << \*it << " ";

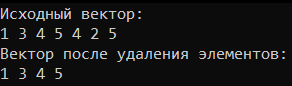
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove\_if**

Пример использования алгоритма remove\_if:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5, 6, 7 };

// Удаляем все четные числа

vec.erase(std::remove\_if(vec.begin(), vec.end(), [](int num) {

return num % 2 == 0;

}), vec.end());

std::cout << "Вектор после удаления четных чисел:\n";

for (const auto& num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove\_copy**

Пример использования алгоритма remove\_copy:

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5, 6, 7 };

std::vector<int> result;

// Копируем все числа, кроме 5, в новый вектор

std::remove\_copy(vec.begin(), vec.end(), std::back\_inserter(result), 5);

std::cout << "Результирующий вектор:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove\_copy\_if**

Пример использования алгоритма remove\_copy\_if:

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5, 6, 7 };

std::vector<int> result;

// Копируем все четные числа в новый вектор

std::remove\_copy\_if(vec.begin(), vec.end(), std::back\_inserter(result), [](int num) { return num % 2 == 0; });

std::cout << "Результирующий вектор:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **copy**

Пример использования алгоритма copy:

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5 };

std::vector<int> result(vec.size());

// Копируем элементы из исходного вектора в новый вектор

std::copy(vec.begin(), vec.end(), result.begin());

std::cout << "Результирующий вектор:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **copy\_backward**

Пример использования алгоритма copy\_backward:

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec = { 1, 2, 3, 4, 5 };

std::vector<int> result(vec.size());

// Копируем элементы из исходного вектора в новый вектор в обратном порядке

std::copy\_backward(vec.begin(), vec.end(), result.end());

std::cout << "Результирующий вектор:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **fill**

Пример использования алгоритма fill:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec(5);

// Заполняем весь вектор значением 42

std::fill(vec.begin(), vec.end(), 42);

std::cout << "Вектор после заполнения:\n";

for (const auto& num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **fill\_n**

Пример использования алгоритма fill\_n:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec(5);

// Заполняем первые 4 элемента вектора значением 42

std::fill\_n(vec.begin(), 4, 42);

std::cout << "Вектор после заполнения:\n";

for (const auto& num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **generate**

Пример использования алгоритма generate:

#include <iostream>

#include <vector>

#include <algorithm>

int generateValue() {

static int value = 0;

return value++;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::vector<int> vec(5);

// Генерируем значения для каждого элемента вектора с помощью функции generateValue

std::generate(vec.begin(), vec.end(), generateValue);

std::cout << "Вектор после генерации значений:\n";

for (const auto& num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **generate\_n**

Пример использования алгоритма generate\_n:

#include <iostream>

#include <vector>

#include <algorithm>

int generateValue() {

static int value = 0;

return value++;

}

int main() {

std::vector<int> vec(5);

std::generate\_n(vec.begin(), 5, generateValue);

std::cout << "Generated vector: ";

for (const auto& element : vec) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partition**

Пример использования алгоритма partition:

#include <iostream>

#include <vector>

#include <algorithm>

bool isEven(int number) {

return number % 2 == 0;

}

int main() {

std::vector<int> vec{ 1, 2, 3, 4, 5, 6, 7, 8 };

std::partition(vec.begin(), vec.end(), isEven);

std::cout << "Partitioned vector: ";

for (const auto& element : vec) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **random\_shuffle**

Пример использования алгоритма random\_shuffle:

#include <iostream>

#include <vector>

#include <algorithm>

#include <random>

int main() {

std::vector<int> vec{ 1, 2, 3, 4, 5 };

std::random\_device rd;

std::mt19937 gen(rd());

std::shuffle(vec.begin(), vec.end(), gen);

std::cout << "Shuffled vector: ";

for (const auto& element : vec) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **reverse**

Пример использования алгоритма reverse:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 2, 3, 4, 5 };

std::reverse(vec.begin(), vec.end());

std::cout << "Reversed vector: ";

for (const auto& element : vec) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **reverse\_copy**

Пример использования алгоритма reverse­\_copy:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 2, 3, 4, 5 };

std::vector<int> reversedVec(vec.size());

std::reverse\_copy(vec.begin(), vec.end(), reversedVec.begin());

std::cout << "Reversed copy of vector: ";

for (const auto& element : reversedVec) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **rotate**

Пример использования алгоритма rotate:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 2, 3, 4, 5 };

std::rotate(vec.begin(), vec.begin() + 3, vec.end());

std::cout << "Rotated vector: ";

for (const auto& element : vec) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **rotate\_copy**

Пример использования алгоритма rotate\_copy:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 2, 3, 4, 5 };

std::vector<int> rotatedVec(vec.size());

std::rotate\_copy(vec.begin(), vec.begin() + 3, vec.end(), rotatedVec.begin());

std::cout << "Original vector: ";

for (const auto& element : vec) {

std::cout << element << " ";

}

std::cout << std::endl;

std::cout << "Rotated vector: ";

for (const auto& element : rotatedVec) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **stable\_partition**

Пример использования алгоритма stable\_partition:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 2, 3, 4, 5 };

auto isEven = [](int num) { return num % 2 == 0; };

std::stable\_partition(vec.begin(), vec.end(), isEven);

std::cout << "Partitioned vector: ";

for (const auto& element : vec) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 2, 3, 4, 5 };

std::vector<int> result(vec.size());

std::transform(vec.begin(), vec.end(), result.begin(), [](int num) {

return num \* 2;

});

std::cout << "Transformed vector: ";

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **unique**

Пример использования алгоритма unique:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 2, 2, 3, 3, 3, 4, 5 };

auto last = std::unique(vec.begin(), vec.end());

std::cout << "Unique elements in vector: ";

for (auto it = vec.begin(); it != last; ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **unique\_copy**

Пример использования алгоритма unique\_copy:

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

int main() {

std::vector<int> vec{ 1, 2, 2, 3, 3, 3, 4, 5 };

std::vector<int> result(vec.size());

auto last = std::unique\_copy(vec.begin(), vec.end(), result.begin());

std::cout << "Unique elements in vector: ";

for (auto it = result.begin(); it != last; ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **inplace\_merge**

Пример использования алгоритма inplace\_merge:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec1{ 1, 3, 5, 7 };

std::vector<int> vec2{ 2, 4, 6, 8 };

std::cout << "Before merge: ";

for (const auto& element : vec1) {

std::cout << element << " ";

}

for (const auto& element : vec2) {

std::cout << element << " ";

}

std::cout << std::endl;

std::inplace\_merge(vec1.begin(), vec1.end(), vec1.end() + vec2.size());

std::cout << "After merge: ";

for (const auto& element : vec1) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **is\_sorted**

Пример использования алгоритма is\_sorted:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 2, 3, 5, 4 };

if (std::is\_sorted(vec.begin(), vec.end())) {

std::cout << "The vector is sorted." << std::endl;

}

else {

std::cout << "The vector is not sorted." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **lexicographical\_compare**

Пример использования алгоритма lexicographical\_copmare:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec1{ 1, 2, 3, 4, 5 };

std::vector<int> vec2{ 1, 2, 3, 5, 4 };

if (std::lexicographical\_compare(vec1.begin(), vec1.end(), vec2.begin(), vec2.end())) {

std::cout << "vec1 is lexicographically less than vec2." << std::endl;

}

else {

std::cout << "vec1 is lexicographically greater or equal to vec2." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **merge**

Пример использования алгоритма merge:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec1{ 1, 3, 5 };

std::vector<int> vec2{ 2, 4, 6 };

std::vector<int> merged;

merged.reserve(vec1.size() + vec2.size());

std::merge(vec1.begin(), vec1.end(), vec2.begin(), vec2.end(), std::back\_inserter(merged));

std::cout << "Merged vector: ";

for (int num : merged) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **nth\_element**

Пример использования алгоритма nth\_element:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 9, 7, 5, 3, 1, 8 };

std::nth\_element(vec.begin(), vec.begin() + 3, vec.end());

std::cout << "Partial sorted vector: ";

for (int num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partial\_sort**

Пример использования алгоритма partial\_sort:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 9, 7, 5, 3, 1, 8 };

std::partial\_sort(vec.begin(), vec.begin() + 3, vec.end());

std::cout << "Partial sorted vector: ";

for (int num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partial\_sort\_copy**

Пример использования алгоритма partial\_sort\_copy:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 9, 7, 5, 3, 1, 8 };

std::vector<int> result(3);

std::partial\_sort\_copy(vec.begin(), vec.end(), result.begin(), result.end());

std::cout << "Partial sorted copy: ";

for (int num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **sort**

Пример использования алгоритма sort:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 9, 7, 5, 3, 1, 8 };

std::sort(vec.begin(), vec.end());

std::cout << "Sorted vector: ";

for (int num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **stable\_sort**

Пример использования алгоритма stable\_sort:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 9, 7, 5, 3, 1, 8 };

std::stable\_sort(vec.begin(), vec.end());

std::cout << "Sorted vector: ";

for (int num : vec) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **binary\_search**

Пример использования алгоритма binary\_search:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 3, 5, 7, 9, 11 };

int target = 7;

bool found = std::binary\_search(vec.begin(), vec.end(), target);

if (found) {

std::cout << "Value " << target << " found in the vector." << std::endl;

}

else {

std::cout << "Value " << target << " not found in the vector." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **equal\_range**

Пример использования алгоритма equal\_range:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 2, 3, 4, 4, 5, 6 };

auto range = std::equal\_range(vec.begin(), vec.end(), 4);

std::cout << "Range of elements equal to 4 in the vector: [";

for (auto it = range.first; it != range.second; ++it) {

std::cout << \*it << " ";

}

std::cout << "]" << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **lower\_bound**

Пример использования алгоритма lower\_bound:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 2, 3, 4, 4, 5, 6 };

auto it = std::lower\_bound(vec.begin(), vec.end(), 4);

if (it != vec.end()) {

std::cout << "First element not less than 4 in the vector: " << \*it << std::endl;

}

else {

std::cout << "No element found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **upper\_bound**

Пример использования алгоритма upper\_bound:

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

std::vector<int> vec{ 1, 2, 3, 4, 4, 5, 6 };

auto it = std::upper\_bound(vec.begin(), vec.end(), 4);

if (it != vec.end()) {

std::cout << "First element greater than 4 in the vector: " << \*it << std::endl;

}

else {

std::cout << "No element found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

## **Deque**

#### **adjacent\_find**

Пример использования алгоритма adjacent\_find:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 3, 4, 5 };

auto result = std::adjacent\_find(deq.begin(), deq.end());

if (result != deq.end()) {

std::cout << "Найдена пара соседних элементов, равных друг другу: "

<< \*result << " и " << \*(result + 1) << std::endl;

}

else {

std::cout << "Соседние элементы, равные друг другу, не найдены." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **count**

Пример использования алгоритма count:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 3, 4, 5 };

int value = 3;

int count = std::count(deq.begin(), deq.end(), value);

std::cout << "Количество вхождений элемента " << value << " в деке: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **count\_if**

Пример использования алгоритма coun\_if:

#include <iostream>

#include <deque>

#include <algorithm>

bool isOdd(int num) {

return num % 2 != 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5, 6 };

int count = std::count\_if(deq.begin(), deq.end(), isOdd);

std::cout << "Количество нечетных элементов в деке: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **equal**

Пример использования алгоритма equal:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq1 = { 1, 2, 3, 4, 5 };

std::deque<int> deq2 = { 1, 2, 3, 4, 5 };

bool isEqual = std::equal(deq1.begin(), deq1.end(), deq2.begin());

std::cout << "Деки deq1 и deq2 " << (isEqual ? "равны" : "не равны") << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find**

Пример использования алгоритма find:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5 };

auto it = std::find(deq.begin(), deq.end(), 3);

if (it != deq.end()) {

std::cout << "Элемент 3 найден в деке deq." << std::endl;

}

else {

std::cout << "Элемент 3 не найден в деке deq." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_if**

Пример использования алгоритма find\_if:

#include <iostream>

#include <deque>

#include <algorithm>

bool isOdd(int num) {

return num % 2 != 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5 };

auto it = std::find\_if(deq.begin(), deq.end(), isOdd);

if (it != deq.end()) {

std::cout << "Первое нечетное число в деке deq: " << \*it << std::endl;

}

else {

std::cout << "Нечетное число не найдено в деке deq." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_end**

Пример использования алгоритма find\_end:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5, 3, 4, 5, 3, 4 };

std::deque<int> subDeq = { 3, 4 };

auto it = std::find\_end(deq.begin(), deq.end(), subDeq.begin(), subDeq.end());

if (it != deq.end()) {

std::cout << "Поддек найден в деке deq, начиная с индекса: " << std::distance(deq.begin(), it) << std::endl;

}

else {

std::cout << "Поддек не найден в деке deq." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_first\_of**

Пример использования алгоритма find\_first\_of:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

std::deque<int> searchValues = { 4, 8, 12 };

auto it = std::find\_first\_of(deq.begin(), deq.end(), searchValues.begin(), searchValues.end());

if (it != deq.end()) {

std::cout << "Первое вхождение одного из искомых значений в деке deq: " << \*it << std::endl;

}

else {

std::cout << "Искомые значения не найдены в деке deq." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **for\_each**

Пример использования алгоритма for\_each:

#include <iostream>

#include <deque>

#include <algorithm>

void printDouble(int num) {

std::cout << num \* 2 << " ";

}

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5 };

std::cout << "Исходная дека: ";

for (int num : deq) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Удвоенные элементы деки: ";

std::for\_each(deq.begin(), deq.end(), printDouble);

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **mismatch**

Пример использования алгоритма mismatch:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq1 = { 1, 2, 3, 4, 5 };

std::deque<int> deq2 = { 1, 2, 3, 6, 5 };

auto pair = std::mismatch(deq1.begin(), deq1.end(), deq2.begin());

if (pair.first == deq1.end()) {

std::cout << "Все элементы деков совпадают." << std::endl;

}

else {

std::cout << "Первое несовпадение: " << \*pair.first << " и " << \*pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **search**

Пример использования алгоритма search:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5, 6, 7, 8 };

std::deque<int> pattern = { 4, 5, 6 };

auto result = std::search(deq.begin(), deq.end(), pattern.begin(), pattern.end());

if (result != deq.end()) {

std::cout << "Найдено совпадение начиная с позиции " << std::distance(deq.begin(), result) << std::endl;

}

else {

std::cout << "Совпадение не найдено." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **search\_n**

Пример использования алгоритма search\_n:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 4, 4, 5, 6 };

int value = 4;

int count = 3;

auto result = std::search\_n(deq.begin(), deq.end(), count, value);

if (result != deq.end()) {

std::cout << "Найдено совпадение " << count << " элементов со значением " << value << " начиная с позиции " << std::distance(deq.begin(), result) << std::endl;

}

else {

std::cout << "Совпадение не найдено." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **max\_element**

Пример использования алгоритма max\_element:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 3, 1, 4, 1, 5 };

int maxValue = \*std::max\_element(deq.begin(), deq.end());

std::cout << "Максимальное значение: " << maxValue << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **min\_element**

Пример использования алгоритма min\_element:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 3, 1, 4, 1, 5 };

int minValue = \*std::min\_element(deq.begin(), deq.end());

std::cout << "Минимальное значение: " << minValue << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **accumulate**

Пример использования алгоритма accumulate:

#include <iostream>

#include <deque>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5 };

int sum = std::accumulate(deq.begin(), deq.end(), 0);

std::cout << "Сумма элементов: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **adjacent\_difference**

Пример использования алгоритма adjacent\_difference:

#include <iostream>

#include <deque>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 3, 5, 7, 9 };

std::deque<int> diff(deq.size());

std::adjacent\_difference(deq.begin(), deq.end(), diff.begin());

std::cout << "Разности между соседними элементами: ";

for (const auto& num : diff) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **inner\_product**

Пример использования алгоритма inner\_product:

#include <iostream>

#include <deque>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq1 = { 1, 2, 3, 4, 5 };

std::deque<int> deq2 = { 6, 7, 8, 9, 10 };

int result = std::inner\_product(deq1.begin(), deq1.end(), deq2.begin(), 0);

std::cout << "Внутреннее произведение: " << result << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partial\_sum**

Пример использования алгоритма partial\_sum:

#include <iostream>

#include <deque>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5 };

std::deque<int> result(deq.size());

std::partial\_sum(deq.begin(), deq.end(), result.begin());

std::cout << "Частичные суммы: ";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **swap**

Пример использования алгоритма swap:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq1 = { 1, 2, 3, 4, 5 };

std::deque<int> deq2 = { 6, 7, 8, 9, 10 };

std::cout << "До обмена:\n";

std::cout << "deq1: ";

for (const auto& num : deq1) {

std::cout << num << " ";

}

std::cout << "\ndeq2: ";

for (const auto& num : deq2) {

std::cout << num << " ";

}

std::cout << std::endl;

std::swap(deq1, deq2);

std::cout << "После обмена:\n";

std::cout << "deq1: ";

for (const auto& num : deq1) {

std::cout << num << " ";

}

std::cout << "\ndeq2: ";

for (const auto& num : deq2) {

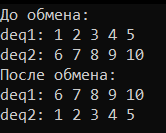
std::cout << num << " ";

}

std::cout << std::endl;

return 0;

} Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **iter\_swap**

Пример использования алгоритма iter\_swap:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5 };

auto it1 = deq.begin();

auto it2 = deq.begin() + 3;

std::cout << "До обмена:\n";

std::cout << "deq: ";

for (const auto& num : deq) {

std::cout << num << " ";

}

std::cout << std::endl;

std::iter\_swap(it1, it2);

std::cout << "После обмена:\n";

std::cout << "deq: ";

for (const auto& num : deq) {

std::cout << num << " ";

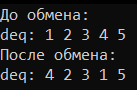
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **swap\_ranges**

Пример использования алгоритма swap\_ranges:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq1 = { 1, 2, 3, 4, 5 };

std::deque<int> deq2 = { 6, 7, 8, 9, 10 };

std::cout << "До обмена:\n";

std::cout << "deq1: ";

for (const auto& num : deq1) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "deq2: ";

for (const auto& num : deq2) {

std::cout << num << " ";

}

std::cout << std::endl;

std::swap\_ranges(deq1.begin(), deq1.end(), deq2.begin());

std::cout << "После обмена:\n";

std::cout << "deq1: ";

for (const auto& num : deq1) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "deq2: ";

for (const auto& num : deq2) {

std::cout << num << " ";

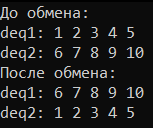
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace**

Пример использования алгоритма replace:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5, 3 };

int oldValue = 3;

int newValue = 9;

std::cout << "До замены:\n";

for (const auto& num : deq) {

std::cout << num << " ";

}

std::cout << std::endl;

std::replace(deq.begin(), deq.end(), oldValue, newValue);

std::cout << "После замены:\n";

for (const auto& num : deq) {

std::cout << num << " ";

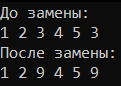
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_if**

Пример использования алгоритма replace\_if:

#include <iostream>

#include <deque>

#include <algorithm>

bool isOdd(int num) {

return num % 2 != 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5, 6 };

int newValue = 0;

std::cout << "До замены:\n";

for (const auto& num : deq) {

std::cout << num << " ";

}

std::cout << std::endl;

std::replace\_if(deq.begin(), deq.end(), isOdd, newValue);

std::cout << "После замены:\n";

for (const auto& num : deq) {

std::cout << num << " ";

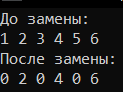
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_copy**

Пример использования алгоритма replace\_copy:

#include <iostream>

#include <deque>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5, 6 };

std::deque<int> newDeq;

int oldValue = 2;

int newValue = 0;

std::replace\_copy(deq.begin(), deq.end(), std::back\_inserter(newDeq), oldValue, newValue);

std::cout << "Исходная двусторонняя очередь:\n";

for (const auto& num : deq) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Новая двусторонняя очередь после замены:\n";

for (const auto& num : newDeq) {

std::cout << num << " ";

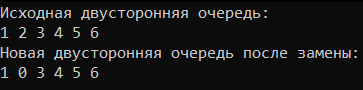
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_copy\_if**

Пример использования алгоритма replace\_copy\_if:

#include <iostream>

#include <deque>

#include <algorithm>

#include <iterator>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5, 6 };

std::deque<int> newDeq;

int newValue = 0;

std::replace\_copy\_if(deq.begin(), deq.end(), std::back\_inserter(newDeq), isEven, newValue);

std::cout << "Исходная двусторонняя очередь:\n";

for (const auto& num : deq) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Новая двусторонняя очередь после замены:\n";

for (const auto& num : newDeq) {

std::cout << num << " ";

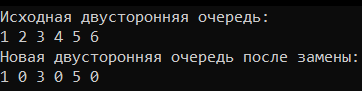
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove**

Пример использования алгоритма remove:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 2, 4, 2, 5 };

int valueToRemove = 2;

auto newEnd = std::remove(deq.begin(), deq.end(), valueToRemove);

std::cout << "Исходная двусторонняя очередь:\n";

for (const auto& num : deq) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Двусторонняя очередь после удаления элементов:\n";

for (auto it = deq.begin(); it != newEnd; ++it) {

std::cout << \*it << " ";

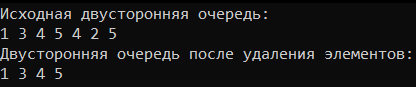
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove\_if**

Пример использования алгоритма remove\_if:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5, 6, 7 };

// Удаляем все числа больше 5

deq.erase(std::remove\_if(deq.begin(), deq.end(), [](int num) {

return num > 5;

}), deq.end());

std::cout << "Двусторонняя очередь после удаления чисел больше 5:\n";

for (const auto& num : deq) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove\_copy**

Пример использования алгоритма remove\_copy:

#include <iostream>

#include <deque>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5, 6, 7 };

std::deque<int> result;

// Копируем все числа, кроме 2, в новую двустороннюю очередь

std::remove\_copy(deq.begin(), deq.end(), std::back\_inserter(result), 2);

std::cout << "Результирующая двусторонняя очередь:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove\_copy\_if**

Пример использования алгоритма remove\_copy\_if:

#include <iostream>

#include <deque>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5, 6, 7 };

std::deque<int> result;

// Копируем все нечетные числа в новую двустороннюю очередь

std::remove\_copy\_if(deq.begin(), deq.end(), std::back\_inserter(result), [](int num) { return num % 2 != 0; });

std::cout << "Результирующая двусторонняя очередь:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **copy**

Пример использования алгоритма copy:

#include <iostream>

#include <deque>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5 };

std::deque<int> result(deq.size());

// Копируем элементы из исходной двусторонней очереди в новую двустороннюю очередь

std::copy(deq.begin(), deq.end(), result.begin());

std::cout << "Результирующая двусторонняя очередь:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **copy\_backward**

Пример использования алгоритма copy\_backward:

#include <iostream>

#include <deque>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq = { 1, 2, 3, 4, 5 };

std::deque<int> result(deq.size());

// Копируем элементы из исходной двусторонней очереди в новую двустороннюю очередь в обратном порядке

std::copy\_backward(deq.begin(), deq.end(), result.end());

std::cout << "Результирующая двусторонняя очередь:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **fill**

Пример использования алгоритма fill:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq(5);

// Заполняем всю двустороннюю очередь значением 42

std::fill(deq.begin(), deq.end(), 42);

std::cout << "Двусторонняя очередь после заполнения:\n";

for (const auto& num : deq) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **fill\_n**

Пример использования алгоритма fill\_n:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq(5);

// Заполняем первые 2 элемента двусторонней очереди значением 42

std::fill\_n(deq.begin(), 2, 42);

std::cout << "Двусторонняя очередь после заполнения:\n";

for (const auto& num : deq) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **generate**

Пример использования алгоритма generate:

#include <iostream>

#include <deque>

#include <algorithm>

int generateValue() {

static int value = 0;

return value++;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::deque<int> deq(5);

// Генерируем значения для каждого элемента двусторонней очереди с помощью функции generateValue

std::generate(deq.begin(), deq.end(), generateValue);

std::cout << "Двусторонняя очередь после генерации значений:\n";

for (const auto& num : deq) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **generate\_n**

Пример использования алгоритма generate\_n:

#include <iostream>

#include <deque>

#include <algorithm>

int generateValue() {

static int value = 0;

return value++;

}

int main() {

std::deque<int> dq(5);

std::generate\_n(dq.begin(), 5, generateValue);

std::cout << "Generated deque: ";

for (const auto& element : dq) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partition**

Пример использования алгоритма partition:

#include <iostream>

#include <deque>

#include <algorithm>

bool isEven(int number) {

return number % 2 == 0;

}

int main() {

std::deque<int> dq{ 1, 2, 3, 4, 5, 6, 7, 8 };

std::partition(dq.begin(), dq.end(), isEven);

std::cout << "Partitioned deque: ";

for (const auto& element : dq) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **random\_shuffle**

Пример использования алгоритма random\_shuffle:

#include <iostream>

#include <deque>

#include <algorithm>

#include <random>

int main() {

std::deque<int> dq{ 1, 2, 3, 4, 5 };

std::random\_device rd;

std::mt19937 gen(rd());

std::shuffle(dq.begin(), dq.end(), gen);

std::cout << "Shuffled deque: ";

for (const auto& element : dq) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **reverse**

Пример использования алгоритма reverse:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> dq{ 1, 2, 3, 4, 5 };

std::reverse(dq.begin(), dq.end());

std::cout << "Reversed deque: ";

for (const auto& element : dq) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **reverse\_copy**

Пример использования алгоритма reverse­\_copy:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> dq{ 1, 2, 3, 4, 5 };

std::deque<int> reversedDq(dq.size());

std::reverse\_copy(dq.begin(), dq.end(), reversedDq.begin());

std::cout << "Reversed copy of deque: ";

for (const auto& element : reversedDq) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **rotate**

Пример использования алгоритма rotate:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> dq{ 1, 2, 3, 4, 5 };

std::rotate(dq.begin(), dq.begin() + 1, dq.end());

std::cout << "Rotated deque: ";

for (const auto& element : dq) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **rotate\_copy**

Пример использования алгоритма rotate\_copy:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> dq{ 1, 2, 3, 4, 5 };

std::deque<int> rotatedDq(dq.size());

std::rotate\_copy(dq.begin(), dq.begin() + 1, dq.end(), rotatedDq.begin());

std::cout << "Original deque: ";

for (const auto& element : dq) {

std::cout << element << " ";

}

std::cout << std::endl;

std::cout << "Rotated deque: ";

for (const auto& element : rotatedDq) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **stable\_partition**

Пример использования алгоритма stable\_partition:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> dq{ 1, 2, 3, 4, 5 };

auto isEven = [](int num) { return num % 2 == 0; };

std::stable\_partition(dq.begin(), dq.end(), isEven);

std::cout << "Partitioned deque: ";

for (const auto& element : dq) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> dq{ 1, 2, 3, 4, 5 };

std::deque<int> result(dq.size());

std::transform(dq.begin(), dq.end(), result.begin(), [](int num) {

return num \* 2;

});

std::cout << "Transformed deque: ";

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **unique**

Пример использования алгоритма unique:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> dq{ 1, 2, 2, 3, 3, 3, 4, 5 };

auto last = std::unique(dq.begin(), dq.end());

std::cout << "Unique elements in deque: ";

for (auto it = dq.begin(); it != last; ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **unique\_copy**

Пример использования алгоритма unique\_copy:

#include <iostream>

#include <deque>

#include <algorithm>

#include <iterator>

int main() {

std::deque<int> dq{ 1, 2, 2, 3, 3, 3, 4, 5 };

std::deque<int> result(dq.size());

auto last = std::unique\_copy(dq.begin(), dq.end(), result.begin());

std::cout << "Unique elements in deque: ";

for (auto it = result.begin(); it != last; ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **inplace\_merge**

Пример использования алгоритма inplace\_merge:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> dq1{ 1, 3, 5, 7 };

std::deque<int> dq2{ 2, 4, 6, 8 };

std::cout << "Before merge: ";

for (const auto& element : dq1) {

std::cout << element << " ";

}

for (const auto& element : dq2) {

std::cout << element << " ";

}

std::cout << std::endl;

std::inplace\_merge(dq1.begin(), dq1.end(), dq1.end() + dq2.size());

std::cout << "After merge: ";

for (const auto& element : dq1) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **is\_sorted**

Пример использования алгоритма is\_sorted:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> dq{ 1, 2, 3, 4, 5 };

if (std::is\_sorted(dq.begin(), dq.end())) {

std::cout << "The deque is sorted." << std::endl;

}

else {

std::cout << "The deque is not sorted." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **lexicographical\_compare**

Пример использования алгоритма lexicographical\_copmare:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> dq1{ 1, 2, 3, 4, 5 };

std::deque<int> dq2{ 1, 2, 3, 5, 4 };

if (std::lexicographical\_compare(dq1.begin(), dq1.end(), dq2.begin(), dq2.end())) {

std::cout << "dq1 is lexicographically less than dq2." << std::endl;

}

else {

std::cout << "dq1 is lexicographically greater or equal to dq2." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **nth\_element**

Пример использования алгоритма nth\_element:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> deque{ 9, 7, 5, 3, 1, 8 };

std::nth\_element(deque.begin(), deque.begin() + 3, deque.end());

std::cout << "Partial sorted deque: ";

for (int num : deque) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partial\_sort**

Пример использования алгоритма partial\_sort:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> deque{ 9, 7, 5, 3, 1, 8 };

std::partial\_sort(deque.begin(), deque.begin() + 3, deque.end());

std::cout << "Partial sorted deque: ";

for (int num : deque) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partial\_sort\_copy**

Пример использования алгоритма partial\_sort\_copy:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> deque{ 9, 7, 5, 3, 1, 8 };

std::deque<int> result(3);

std::partial\_sort\_copy(deque.begin(), deque.end(), result.begin(), result.end());

std::cout << "Partial sorted copy: ";

for (int num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **sort**

Пример использования алгоритма sort:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> deque{ 9, 7, 5, 3, 1, 8 };

std::sort(deque.begin(), deque.end());

std::cout << "Sorted deque: ";

for (int num : deque) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **stable\_sort**

Пример использования алгоритма stable\_sort:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> deque{ 9, 7, 5, 3, 1, 8 };

std::stable\_sort(deque.begin(), deque.end());

std::cout << "Sorted deque: ";

for (int num : deque) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **binary\_search**

Пример использования алгоритма binary\_search:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> deque{ 1, 3, 5, 7, 9, 11 };

int target = 7;

bool found = std::binary\_search(deque.begin(), deque.end(), target);

if (found) {

std::cout << "Value " << target << " found in the deque." << std::endl;

}

else {

std::cout << "Value " << target << " not found in the deque." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **equal\_range**

Пример использования алгоритма equal\_range:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> deque{ 1, 2, 3, 4, 4, 5, 6 };

auto range = std::equal\_range(deque.begin(), deque.end(), 4);

std::cout << "Range of elements equal to 4 in the deque: [";

for (auto it = range.first; it != range.second; ++it) {

std::cout << \*it << " ";

}

std::cout << "]" << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **lower\_bound**

Пример использования алгоритма lower\_bound:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> deque{ 1, 2, 3, 4, 4, 5, 6 };

auto it = std::lower\_bound(deque.begin(), deque.end(), 4);

if (it != deque.end()) {

std::cout << "First element not less than 4 in the deque: " << \*it << std::endl;

}

else {

std::cout << "No element found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **upper\_bound**

Пример использования алгоритма upper\_bound:

#include <iostream>

#include <deque>

#include <algorithm>

int main() {

std::deque<int> deque{ 1, 2, 3, 4, 4, 5, 6 };

auto it = std::upper\_bound(deque.begin(), deque.end(), 4);

if (it != deque.end()) {

std::cout << "First element greater than 4 in the deque: " << \*it << std::endl;

}

else {

std::cout << "No element found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

## **Forward\_list**

#### **find**

Пример использования алгоритма find:

#include <iostream>

#include <forward\_list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::forward\_list<int> flist = { 1, 2, 3, 4, 5 };

auto it = std::find(flist.begin(), flist.end(), 3);

if (it != flist.end()) {

std::cout << "Элемент 3 найден в односвязном списке flist." << std::endl;

}

else {

std::cout << "Элемент 3 не найден в односвязном списке flist." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_if**

Пример использования алгоритма find\_if:

#include <iostream>

#include <forward\_list>

#include <algorithm>

bool isPrime(int num) {

if (num < 2) return false;

for (int i = 2; i \* i <= num; ++i) {

if (num % i == 0) return false;

}

return true;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::forward\_list<int> flist = { 1, 2, 3, 4, 5 };

auto it = std::find\_if(flist.begin(), flist.end(), isPrime);

if (it != flist.end()) {

std::cout << "Первое простое число в односвязном списке flist: " << \*it << std::endl;

}

else {

std::cout << "Простое число не найдено в односвязном списке flist." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **for\_each**

Пример использования алгоритма for\_each:

#include <iostream>

#include <forward\_list>

#include <algorithm>

void printTriple(int num) {

std::cout << num \* 3 << " ";

}

int main() {

setlocale(LC\_ALL, "Russian");

std::forward\_list<int> flist = { 1, 2, 3, 4, 5 };

std::cout << "Исходный однонаправленный список: ";

for (int num : flist) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Троенные элементы списка: ";

std::for\_each(flist.begin(), flist.end(), printTriple);

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **mismatch**

Пример использования алгоритма mismatch:

#include <iostream>

#include <forward\_list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::forward\_list<int> flist1 = { 1, 2, 3, 4, 5 };

std::forward\_list<int> flist2 = { 1, 2, 3, 6, 5 };

auto pair = std::mismatch(flist1.begin(), flist1.end(), flist2.begin());

if (pair.first == flist1.end()) {

std::cout << "Все элементы однонаправленных списков совпадают." << std::endl;

}

else {

std::cout << "Первое несовпадение: " << \*pair.first << " и " << \*pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **copy**

Пример использования алгоритма copy:

#include <iostream>

#include <forward\_list>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::forward\_list<int> forwardList = { 1, 2, 3, 4, 5 };

std::forward\_list<int> result;

// Копируем элементы из исходного односвязного списка в новый односвязный список

std::copy(forwardList.begin(), forwardList.end(), std::front\_inserter(result));

std::cout << "Результирующий односвязный список:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **generate**

Пример использования алгоритма generate:

#include <iostream>

#include <forward\_list>

#include <algorithm>

int generateValue() {

static int value = 0;

return value++;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::forward\_list<int> flist(5);

// Генерируем значения для каждого элемента односвязного списка с помощью функции generateValue

std::generate(flist.begin(), flist.end(), generateValue);

std::cout << "Односвязный список после генерации значений:\n";

for (const auto& num : flist) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **generate\_n**

Пример использования алгоритма generate\_n:

#include <iostream>

#include <forward\_list>

#include <algorithm>

int generateValue() {

static int value = 0;

return value++;

}

int main() {

std::forward\_list<int> flist(5);

std::generate\_n(flist.begin(), 5, generateValue);

std::cout << "Generated forward list: ";

for (const auto& element : flist) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **reverse**

Пример использования алгоритма reverse:

#include <iostream>

#include <forward\_list>

#include <algorithm>

int main() {

std::forward\_list<int> flist{ 1, 2, 3, 4, 5 };

flist.reverse();

std::cout << "Reversed forward\_list: ";

for (const auto& element : flist) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **stable\_partition**

Пример использования алгоритма stable\_partition:

#include <iostream>

#include <forward\_list>

#include <algorithm>

int main() {

std::forward\_list<int> fwdList{ 1, 2, 3, 4, 5 };

auto isEven = [](int num) { return num % 2 == 0; };

// Используем итераторы перед началом и после конца, чтобы указать границы последовательности

auto beforeBegin = fwdList.before\_begin();

auto end = fwdList.end();

std::stable\_partition(++beforeBegin, end, isEven);

std::cout << "Partitioned forward list: ";

for (const auto& element : fwdList) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <forward\_list>

#include <algorithm>

#include <iterator>

int main() {

std::forward\_list<int> fwdList{ 1, 2, 3, 4, 5 };

std::forward\_list<int> result;

std::transform(fwdList.begin(), fwdList.end(), std::front\_inserter(result), [](int num) {

return num \* 2;

});

std::cout << "Transformed forward list: ";

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **unique**

Пример использования алгоритма unique:

#include <iostream>

#include <forward\_list>

#include <algorithm>

#include <iterator>

int main() {

std::forward\_list<int> fwdList{ 1, 2, 2, 3, 3, 3, 4, 5 };

fwdList.unique();

std::cout << "Unique elements in forward list: ";

for (const auto& element : fwdList) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **unique\_copy**

Пример использования алгоритма unique\_copy:

#include <iostream>

#include <forward\_list>

#include <algorithm>

#include <iterator>

int main() {

std::forward\_list<int> fwdList{ 1, 2, 2, 3, 3, 3, 4, 5 };

std::forward\_list<int> result;

auto it = std::unique\_copy(fwdList.begin(), fwdList.end(), std::front\_inserter(result));

std::cout << "Unique elements in forward list: ";

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **upper\_bound**

Пример использования алгоритма upper\_bound:

#include <iostream>

#include <forward\_list>

#include <algorithm>

int main() {

std::forward\_list<int> forwardList{ 1, 2, 3, 4, 4, 5, 6 };

auto it = std::upper\_bound(forwardList.begin(), forwardList.end(), 4);

if (it != forwardList.end()) {

std::cout << "First element greater than 4 in the forward\_list: " << \*it << std::endl;

}

else {

std::cout << "No element found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

## **List**

#### **adjacent\_find**

Пример использования алгоритма adjacent\_find:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 3, 4, 5 };

auto result = std::adjacent\_find(lst.begin(), lst.end());

if (result != lst.end()) {

std::cout << "Найдена пара соседних элементов, равных друг другу: "

<< \*result << " и " << \*(++result) << std::endl;

}

else {

std::cout << "Соседние элементы, равные друг другу, не найдены." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **count**

Пример использования алгоритма count:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 3, 4, 5 };

int value = 3;

int count = std::count(lst.begin(), lst.end(), value);

std::cout << "Количество вхождений элемента " << value << " в списке: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **count\_if**

Пример использования алгоритма count\_if:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 3, 4, 5 };

int value = 3;

int count = std::count(lst.begin(), lst.end(), value);

std::cout << "Количество вхождений элемента " << value << " в списке: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **equal**

Пример использования алгоритма equal:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst1 = { 1, 2, 3, 4, 5 };

std::list<int> lst2 = { 1, 2, 3, 4, 5 };

bool isEqual = std::equal(lst1.begin(), lst1.end(), lst2.begin());

std::cout << "Списки lst1 и lst2 " << (isEqual ? "равны" : "не равны") << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find**

Пример использования алгоритма find:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5 };

auto it = std::find(lst.begin(), lst.end(), 3);

if (it != lst.end()) {

std::cout << "Элемент 3 найден в списке lst." << std::endl;

}

else {

std::cout << "Элемент 3 не найден в списке lst." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_if**

Пример использования алгоритма find\_if:

#include <iostream>

#include <list>

#include <algorithm>

bool isNegative(int num) {

return num < 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { -2, -1, 0, 1, 2 };

auto it = std::find\_if(lst.begin(), lst.end(), isNegative);

if (it != lst.end()) {

std::cout << "Первое отрицательное число в списке lst: " << \*it << std::endl;

}

else {

std::cout << "Отрицательное число не найдено в списке lst." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_end**

Пример использования алгоритма find\_end:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5, 3, 4, 5, 3, 4 };

std::list<int> subList = { 3, 4 };

auto it = std::find\_end(lst.begin(), lst.end(), subList.begin(), subList.end());

if (it != lst.end()) {

std::cout << "Подсписок найден в списке lst, начиная с индекса: " << std::distance(lst.begin(), it) << std::endl;

}

else {

std::cout << "Подсписок не найден в списке lst." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **find\_first\_of**

Пример использования алгоритма find\_first\_of:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

std::list<int> searchValues = { 4, 8, 12 };

auto it = std::find\_first\_of(lst.begin(), lst.end(), searchValues.begin(), searchValues.end());

if (it != lst.end()) {

std::cout << "Первое вхождение одного из искомых значений в списке lst: " << \*it << std::endl;

}

else {

std::cout << "Искомые значения не найдены в списке lst." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **for\_each**

Пример использования алгоритма for\_each:

#include <iostream>

#include <list>

#include <algorithm>

void printSquare(int num) {

std::cout << num \* num << " ";

}

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5 };

std::cout << "Исходный список: ";

for (int num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Квадраты элементов списка: ";

std::for\_each(lst.begin(), lst.end(), printSquare);

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **mismatch**

Пример использования алгоритма mismatch:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> list1 = { 1, 2, 3, 4, 5 };

std::list<int> list2 = { 1, 2, 3, 6, 5 };

auto pair = std::mismatch(list1.begin(), list1.end(), list2.begin());

if (pair.first == list1.end()) {

std::cout << "Все элементы списков совпадают." << std::endl;

}

else {

std::cout << "Первое несовпадение: " << \*pair.first << " и " << \*pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **search**

Пример использования алгоритма search:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> list = { 1, 2, 3, 4, 5, 6, 7, 8 };

std::list<int> pattern = { 4, 5, 6 };

auto result = std::search(list.begin(), list.end(), pattern.begin(), pattern.end());

if (result != list.end()) {

std::cout << "Найдено совпадение начиная с позиции " << std::distance(list.begin(), result) << std::endl;

}

else {

std::cout << "Совпадение не найдено." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **search\_n**

Пример использования алгоритма search\_n:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 4, 4, 5, 6 };

int value = 4;

int count = 3;

auto result = std::search\_n(lst.begin(), lst.end(), count, value);

if (result != lst.end()) {

std::cout << "Найдено совпадение " << count << " элементов со значением " << value << " начиная с позиции " << std::distance(lst.begin(), result) << std::endl;

}

else {

std::cout << "Совпадение не найдено." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **max\_element**

Пример использования алгоритма max\_element:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 3, 1, 4, 1, 5 };

int maxValue = \*std::max\_element(lst.begin(), lst.end());

std::cout << "Максимальное значение: " << maxValue << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **min\_element**

Пример использования алгоритма min\_element:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 3, 1, 4, 1, 5 };

int minValue = \*std::min\_element(lst.begin(), lst.end());

std::cout << "Минимальное значение: " << minValue << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **accumulate**

Пример использования алгоритма accumulate:

#include <iostream>

#include <list>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5 };

int sum = std::accumulate(lst.begin(), lst.end(), 0);

std::cout << "Сумма элементов: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **adjacent\_difference**

Пример использования алгоритма adjacent\_difference:

#include <iostream>

#include <list>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 3, 5, 7, 9 };

std::list<int> diff;

std::adjacent\_difference(lst.begin(), lst.end(), std::back\_inserter(diff));

std::cout << "Разности между соседними элементами: ";

for (const auto& num : diff) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **inner\_product**

Пример использования алгоритма inner\_product:

#include <iostream>

#include <list>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst1 = { 1, 2, 3, 4, 5 };

std::list<int> lst2 = { 6, 7, 8, 9, 10 };

int result = std::inner\_product(lst1.begin(), lst1.end(), lst2.begin(), 0);

std::cout << "Внутреннее произведение: " << result << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partial\_sum**

Пример использования алгоритма partial\_sum:

#include <iostream>

#include <list>

#include <numeric>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5 };

std::list<int> result;

std::partial\_sum(lst.begin(), lst.end(), std::back\_inserter(result));

std::cout << "Частичные суммы: ";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace**

Пример использования алгоритма replace:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5, 3 };

int oldValue = 3;

int newValue = 9;

std::cout << "До замены:\n";

for (const auto& num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

std::replace(lst.begin(), lst.end(), oldValue, newValue);

std::cout << "После замены:\n";

for (const auto& num : lst) {

std::cout << num << " ";

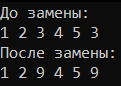
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_if**

Пример использования алгоритма replace\_if:

#include <iostream>

#include <list>

#include <algorithm>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5, 6 };

int newValue = 0;

std::cout << "До замены:\n";

for (const auto& num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

std::replace\_if(lst.begin(), lst.end(), isEven, newValue);

std::cout << "После замены:\n";

for (const auto& num : lst) {

std::cout << num << " ";

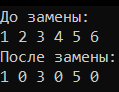
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_copy**

Пример использования алгоритма replace\_copy:

#include <iostream>

#include <list>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5, 6 };

std::list<int> newList;

int oldValue = 2;

int newValue = 0;

std::replace\_copy(lst.begin(), lst.end(), std::back\_inserter(newList), oldValue, newValue);

std::cout << "Исходный список:\n";

for (const auto& num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Новый список после замены:\n";

for (const auto& num : newList) {

std::cout << num << " ";

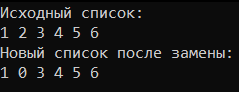
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **replace\_copy\_if**

Пример использования алгоритма replace\_copy\_if:

#include <iostream>

#include <list>

#include <algorithm>

#include <iterator>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5, 6 };

std::list<int> newList;

int newValue = 0;

std::replace\_copy\_if(lst.begin(), lst.end(), std::back\_inserter(newList), isEven, newValue);

std::cout << "Исходный список:\n";

for (const auto& num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Новый список после замены:\n";

for (const auto& num : newList) {

std::cout << num << " ";

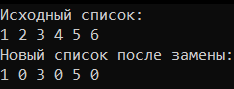
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove**

Пример использования алгоритма remove:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 2, 4, 2, 5 };

int valueToRemove = 2;

auto newEnd = std::remove(lst.begin(), lst.end(), valueToRemove);

std::cout << "Исходный список:\n";

for (const auto& num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Список после удаления элементов:\n";

for (auto it = lst.begin(); it != newEnd; ++it) {

std::cout << \*it << " ";

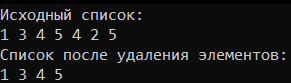
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove\_if**

Пример использования алгоритма remove\_if:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5, 6, 7 };

// Удаляем все числа меньше или равные 3

lst.remove\_if([](int num) {

return num <= 3;

});

std::cout << "Список после удаления чисел меньше или равных 3:\n";

for (const auto& num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove\_copy**

Пример использования алгоритма remove\_copy:

#include <iostream>

#include <list>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5, 6, 7 };

std::list<int> result;

// Копируем все числа, кроме 4, в новый список

std::remove\_copy(lst.begin(), lst.end(), std::back\_inserter(result), 4);

std::cout << "Результирующий список:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **remove\_copy\_if**

Пример использования алгоритма remove\_copy\_if:

#include <iostream>

#include <list>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5, 6, 7 };

std::list<int> result;

// Копируем все нечетные числа в новый список

std::remove\_copy\_if(lst.begin(), lst.end(), std::back\_inserter(result), [](int num) { return num % 2 != 0; });

std::cout << "Результирующий список:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **copy**

Пример использования алгоритма copy:

#include <iostream>

#include <list>

#include <algorithm>

#include <iterator>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst = { 1, 2, 3, 4, 5 };

std::list<int> result(lst.size());

// Копируем элементы из исходного списка в новый список

std::copy(lst.begin(), lst.end(), result.begin());

std::cout << "Результирующий список:\n";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **fill**

Пример использования алгоритма fill:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst(5);

// Заполняем всего списка значением 42

std::fill(lst.begin(), lst.end(), 42);

std::cout << "Список после заполнения:\n";

for (const auto& num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **fill\_n**

Пример использования алгоритма fill\_n:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst(5);

// Заполняем первые 3 элемента списка значением 42

std::fill\_n(lst.begin(), 3, 42);

std::cout << "Список после заполнения:\n";

for (const auto& num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **generate**

Пример использования алгоритма generate:

#include <iostream>

#include <list>

#include <algorithm>

int generateValue() {

static int value = 0;

return value++;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::list<int> lst(5);

// Генерируем значения для каждого элемента списка с помощью функции generateValue

std::generate(lst.begin(), lst.end(), generateValue);

std::cout << "Список после генерации значений:\n";

for (const auto& num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **generate\_n**

Пример использования алгоритма generate\_n:

#include <iostream>

#include <list>

#include <algorithm>

int generateValue() {

static int value = 0;

return value++;

}

int main() {

std::list<int> lst(5);

std::generate\_n(lst.begin(), 5, generateValue);

std::cout << "Generated list: ";

for (const auto& element : lst) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **partition**

Пример использования алгоритма partition:

#include <iostream>

#include <list>

#include <algorithm>

bool isEven(int number) {

return number % 2 == 0;

}

int main() {

std::list<int> lst{ 1, 2, 3, 4, 5, 6, 7, 8 };

std::partition(lst.begin(), lst.end(), isEven);

std::cout << "Partitioned list: ";

for (const auto& element : lst) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **random\_shuffle**

Пример использования алгоритма random\_shuffle:

#include <iostream>

#include <list>

#include <algorithm>

#include <random>

int main() {

std::list<int> lst{ 1, 2, 3, 4, 5 };

std::random\_device rd;

std::mt19937 gen(rd());

std::vector<int> temp(lst.begin(), lst.end());

std::shuffle(temp.begin(), temp.end(), gen);

std::copy(temp.begin(), temp.end(), lst.begin());

std::cout << "Shuffled list: ";

for (const auto& element : lst) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **reverse**

Пример использования алгоритма reverse:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> lst{ 1, 2, 3, 4, 5 };

lst.reverse();

std::cout << "Reversed list: ";

for (const auto& element : lst) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **reverse\_copy**

Пример использования алгоритма reverse­\_copy:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> lst{ 1, 2, 3, 4, 5 };

std::list<int> reversedLst;

std::reverse\_copy(lst.begin(), lst.end(), std::back\_inserter(reversedLst));

std::cout << "Reversed copy of list: ";

for (const auto& element : reversedLst) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **rotate**

Пример использования алгоритма rotate:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> lst{ 1, 2, 3, 4, 5 };

std::rotate(lst.begin(), std::next(lst.begin()), lst.end());

std::cout << "Rotated list: ";

for (const auto& element : lst) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **rotate\_copy**

Пример использования алгоритма rotate\_copy:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> lst{ 1, 2, 3, 4, 5 };

std::list<int> rotatedLst(lst.size());

std::rotate\_copy(lst.begin(), lst.begin()++, lst.end(), rotatedLst.begin());

std::cout << "Original list: ";

for (const auto& element : lst) {

std::cout << element << " ";

}

std::cout << std::endl;

std::cout << "Rotated list: ";

for (const auto& element : rotatedLst) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **stable\_partition**

Пример использования алгоритма stable\_partition:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> lst{ 1, 2, 3, 4, 5 };

auto isEven = [](int num) { return num % 2 == 0; };

std::stable\_partition(lst.begin(), lst.end(), isEven);

std::cout << "Partitioned list: ";

for (const auto& element : lst) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> lst{ 1, 2, 3, 4, 5 };

std::list<int> result;

std::transform(lst.begin(), lst.end(), std::back\_inserter(result), [](int num) {

return num \* 2;

});

std::cout << "Transformed list: ";

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **unique**

Пример использования алгоритма unique:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> lst{ 1, 2, 2, 3, 3, 3, 4, 5 };

lst.unique();

std::cout << "Unique elements in list: ";

for (const auto& element : lst) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **unique\_copy**

Пример использования алгоритма unique\_copy:

#include <iostream>

#include <list>

#include <algorithm>

#include <iterator>

int main() {

std::list<int> lst{ 1, 2, 2, 3, 3, 3, 4, 5 };

std::list<int> result;

auto it = std::unique\_copy(lst.begin(), lst.end(), std::back\_inserter(result));

std::cout << "Unique elements in list: ";

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **inplace\_merge**

Пример использования алгоритма inplace\_merge:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> list1{ 1, 3, 5, 7 };

std::list<int> list2{ 2, 4, 6, 8 };

std::cout << "Before merge: ";

for (const auto& element : list1) {

std::cout << element << " ";

}

for (const auto& element : list2) {

std::cout << element << " ";

}

std::cout << std::endl;

std::inplace\_merge(list1.begin(), list1.end(), list2.end());

std::cout << "After merge: ";

for (const auto& element : list1) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **is\_sorted**

Пример использования алгоритма is\_sorted:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> list{ 1, 2, 3, 5, 4 };

if (std::is\_sorted(list.begin(), list.end())) {

std::cout << "The list is sorted." << std::endl;

}

else {

std::cout << "The list is not sorted." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **lexicographical\_compare**

Пример использования алгоритма lexicographical\_copmare:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> list1{ 1, 2, 3, 4, 5 };

std::list<int> list2{ 1, 2, 3, 5, 4 };

if (std::lexicographical\_compare(list1.begin(), list1.end(), list2.begin(), list2.end())) {

std::cout << "list1 is lexicographically less than list2." << std::endl;

}

else {

std::cout << "list1 is lexicographically greater or equal to list2." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **merge**

Пример использования алгоритма merge:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> list1{ 1, 3, 5 };

std::list<int> list2{ 2, 4, 6 };

std::list<int> merged;

std::merge(list1.begin(), list1.end(), list2.begin(), list2.end(), std::inserter(merged, merged.begin()));

std::cout << "Merged list: ";

for (int num : merged) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **sort**

Пример использования алгоритма sort:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> list{ 9, 7, 5, 3, 1, 8 };

list.sort();

std::cout << "Sorted list: ";

for (int num : list) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **stable\_sort**

Пример использования алгоритма stable\_sort:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> list{ 9, 7, 5, 3, 1, 8 };

list.sort();

std::cout << "Sorted list: ";

for (int num : list) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **binary\_search**

Пример использования алгоритма binary\_search:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> list{ 1, 3, 5, 7, 9, 11 };

int target = 7;

bool found = std::binary\_search(list.begin(), list.end(), target);

if (found) {

std::cout << "Value " << target << " found in the list." << std::endl;

}

else {

std::cout << "Value " << target << " not found in the list." << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **equal\_range**

Пример использования алгоритма equal\_range:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> list{ 1, 2, 3, 4, 4, 5, 6 };

auto range = std::equal\_range(list.begin(), list.end(), 4);

std::cout << "Range of elements equal to 4 in the list: [";

for (auto it = range.first; it != range.second; ++it) {

std::cout << \*it << " ";

}

std::cout << "]" << std::endl;

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **lower\_bound**

Пример использования алгоритма lower\_bound:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> list{ 1, 2, 3, 4, 4, 5, 6 };

auto it = std::lower\_bound(list.begin(), list.end(), 4);

if (it != list.end()) {

std::cout << "First element not less than 4 in the list: " << \*it << std::endl;

}

else {

std::cout << "No element found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

#### **upper\_bound**

Пример использования алгоритма upper\_bound:

#include <iostream>

#include <list>

#include <algorithm>

int main() {

std::list<int> myList{ 1, 2, 3, 4, 4, 5, 6 };

auto it = std::upper\_bound(myList.begin(), myList.end(), 4);

if (it != myList.end()) {

std::cout << "First element greater than 4 in the list: " << \*it << std::endl;

}

else {

std::cout << "No element found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Последовательные контейнеры>](#_Последовательные_контейнеры_1)

# **Ассоциативные контейнеры**

## **Set**

#### **Set**

##### **adjacent\_find**

Пример использования алгоритма adjacent\_find:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::set<int> mySet{ 1, 2, 3, 4, 4, 5 };

auto adjIt = std::adjacent\_find(mySet.begin(), mySet.end());

if (adjIt != mySet.end()) {

std::cout << "Первая пара соседних элементов: " << \*adjIt << " и " << \*(++adjIt) << std::endl;

}

else {

std::cout << "Соседние элементы не найдены." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **count**

Пример использования алгоритма count:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::set<int> mySet{ 1, 2, 3, 4, 4, 5 };

int count = std::count(mySet.begin(), mySet.end(), 4);

std::cout << "Количество элементов со значением 4: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **count\_if**

Пример использования алгоритма count\_if:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::set<int> mySet{ 1, 2, 3, 4, 5 };

int countIf = std::count\_if(mySet.begin(), mySet.end(), [](int num) { return num > 2; });

std::cout << "Количество элементов, больших 2: " << countIf << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **equal**

Пример использования алгоритма equal:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::set<int> set1{ 1, 2, 3, 4, 5 };

std::set<int> set2{ 1, 2, 3, 4, 5 };

bool isEqual = std::equal(set1.begin(), set1.end(), set2.begin());

std::cout << "Множества одинаковы: " << std::boolalpha << isEqual << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find**

Пример использования алгоритма find:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::set<int> numbers = { 1, 2, 3, 4, 5 };

auto it = std::find(numbers.begin(), numbers.end(), 3);

if (it != numbers.end()) {

std::cout << "Number found: " << \*it << std::endl;

}

else {

std::cout << "Number not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find\_if**

Пример использования алгоритма find\_if:

#include <iostream>

#include <set>

#include <algorithm>

bool IsEven(int number) {

return number % 2 == 0;

}

int main() {

std::set<int> numbers = { 1, 2, 3, 4, 5 };

auto it = std::find\_if(numbers.begin(), numbers.end(), IsEven);

if (it != numbers.end()) {

std::cout << "First even number found: " << \*it << std::endl;

}

else {

std::cout << "No even number found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find\_end**

Пример использования алгоритма find\_end:

#include <iostream>

#include <set>

#include <algorithm>

bool IsEven(int number) {

return number % 2 == 0;

}

int main() {

std::set<int> numbers = { 1, 2, 3, 4, 5 };

auto it = std::find\_if(numbers.begin(), numbers.end(), IsEven);

if (it != numbers.end()) {

std::cout << "First even number found: " << \*it << std::endl;

}

else {

std::cout << "No even number found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find\_first\_of**

Пример использования алгоритма find\_first\_of:

#include <iostream>

#include <set>

#include <algorithm>

#include <string>

int main() {

std::set<char> vowels = { 'a', 'e', 'i', 'o', 'u' };

std::string word = "hello";

auto it = std::find\_first\_of(word.begin(), word.end(), vowels.begin(), vowels.end());

if (it != word.end()) {

std::cout << "First vowel found: " << \*it << std::endl;

}

else {

std::cout << "No vowel found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **for\_each**

Пример использования алгоритма for\_each:

#include <iostream>

#include <set>

#include <algorithm>

struct Print {

void operator()(int value) const {

std::cout << value << " ";

}

};

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

std::for\_each(mySet.begin(), mySet.end(), Print());

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **mismatch**

Пример использования алгоритма mismatch:

#include <iostream>

#include <set>

#include <algorithm>

#include <string>

int main() {

std::set<char> set1 = { 'h', 'e', 'l', 'l', 'o' };

std::set<char> set2 = { 'h', 'e', 'l', 'p' };

auto mismatchPair = std::mismatch(set1.begin(), set1.end(), set2.begin());

if (mismatchPair.first != set1.end()) {

std::cout << "Mismatch found at position: " << std::distance(set1.begin(), mismatchPair.first) << std::endl;

}

else {

std::cout << "No mismatch found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **search**

Пример использования алгоритма search:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

std::set<int> subSet = { 3, 4 };

auto it = std::search(mySet.begin(), mySet.end(), subSet.begin(), subSet.end());

if (it != mySet.end()) {

std::cout << "Subsequence found at index: " << std::distance(mySet.begin(), it) << std::endl;

}

else {

std::cout << "Subsequence not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **search\_n**

Пример использования алгоритма search\_n:

#include <iostream>

#include <set>

#include <algorithm>

// Функция, которую мы будем применять к каждому элементу множества

void process\_element(int element) {

std::cout << element << " ";

}

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

// Применяем функцию process\_element к каждому элементу множества

std::for\_each(mySet.begin(), mySet.end(), process\_element);

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **max\_element**

Пример использования алгоритма max\_element:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

auto maxElement = std::max\_element(mySet.begin(), mySet.end());

std::cout << "Max element: " << \*maxElement << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **min\_element**

Пример использования алгоритма min\_element:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

auto minElement = std::min\_element(mySet.begin(), mySet.end());

std::cout << "Min element: " << \*minElement << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **accumulate**

Пример использования алгоритма accumulate:

#include <iostream>

#include <set>

#include <numeric>

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

int sum = std::accumulate(mySet.begin(), mySet.end(), 0);

std::cout << "Sum: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **adjacent\_difference**

Пример использования алгоритма adjacent\_difference:

#include <iostream>

#include <set>

#include <numeric>

int main() {

std::set<int> mySet = { 1, 3, 5, 7, 9 };

std::set<int> result;

std::adjacent\_difference(mySet.begin(), mySet.end(), std::inserter(result, result.begin()));

std::cout << "Result: ";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **inner\_product**

Пример использования алгоритма inner\_product:

#include <iostream>

#include <set>

#include <numeric>

int main() {

std::set<int> set1 = { 1, 2, 3 };

std::set<int> set2 = { 4, 5, 6 };

int result = std::inner\_product(set1.begin(), set1.end(), set2.begin(), 0);

std::cout << "Result: " << result << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **partial\_sum**

Пример использования алгоритма partial\_sum:

#include <iostream>

#include <set>

#include <numeric>

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

std::set<int> result;

std::partial\_sum(mySet.begin(), mySet.end(), std::inserter(result, result.begin()));

std::cout << "Result: ";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **swap**

Пример использования алгоритма swap:

#include <iostream>

#include <set>

int main() {

std::set<int> set1 = { 1, 2, 3 };

std::set<int> set2 = { 4, 5, 6 };

std::swap(set1, set2);

std::cout << "Set 1: ";

for (const auto& num : set1) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Set 2: ";

for (const auto& num : set2) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **replace\_copy**

Пример использования алгоритма replace\_copy:

#include <iostream>

#include <algorithm>

#include <set>

#include <iterator>

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

std::set<int> result;

std::replace\_copy(mySet.begin(), mySet.end(), std::inserter(result, result.begin()), 3, 10);

// Выводим элементы в результате замены

for (const auto& value : result) {

std::cout << value << " ";

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **replace\_copy\_if**

Пример использования алгоритма replace\_copy\_if:

#include <iostream>

#include <algorithm>

#include <set>

#include <iterator>

bool isEven(int value) {

return value % 2 == 0;

}

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

std::set<int> result;

std::replace\_copy\_if(mySet.begin(), mySet.end(), std::inserter(result, result.begin()), isEven, 0);

// Выводим элементы в результате замены

for (const auto& value : result) {

std::cout << value << " ";

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **remove\_copy**

Пример использования алгоритма remove\_copy:

#include <iostream>

#include <set>

#include <vector>

#include <algorithm>

int main() {

std::set<int> mySet{ 1, 2, 3, 4, 5 };

std::vector<int> result;

std::remove\_copy(mySet.begin(), mySet.end(), std::back\_inserter(result), 3);

std::cout << "Elements in result vector after 'remove\_copy': ";

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **remove\_copy\_if**

Пример использования алгоритма remove\_copy\_if:

#include <iostream>

#include <set>

#include <vector>

#include <algorithm>

bool isOdd(int num) {

return num % 2 != 0;

}

int main() {

std::set<int> mySet{ 1, 2, 3, 4, 5 };

std::vector<int> result;

std::remove\_copy\_if(mySet.begin(), mySet.end(), std::back\_inserter(result), isOdd);

std::cout << "Elements in result vector after 'remove\_copy\_if': ";

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **copy**

Пример использования алгоритма copy:

#include <iostream>

#include <set>

#include <algorithm>

#include <vector>

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

std::vector<int> myVector;

std::copy(mySet.begin(), mySet.end(), std::back\_inserter(myVector));

std::cout << "Copied elements: ";

for (int value : myVector) {

std::cout << value << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **copy\_backward**

Пример использования алгоритма copy\_backward:

#include <iostream>

#include <set>

#include <algorithm>

#include <vector>

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

std::vector<int> myVector;

std::copy(mySet.begin(), mySet.end(), std::back\_inserter(myVector));

std::cout << "Copied elements: ";

for (int value : myVector) {

std::cout << value << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <set>

#include <algorithm>

#include <vector>

int square(int number) {

return number \* number;

}

int main() {

std::set<int> mySet{ 1, 2, 3, 4, 5 };

std::vector<int> transformedSet;

std::transform(mySet.begin(), mySet.end(), std::back\_inserter(transformedSet), [](int number) {

return number \* number;

});

for (const auto& element : transformedSet) {

std::cout << element << " ";

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **unique\_copy**

Пример использования алгоритма unique\_copy:

#include <iostream>

#include <set>

#include <algorithm>

#include <vector>

int main() {

std::set<int> mySet{ 1, 2, 2, 3, 4, 4, 5 };

std::vector<int> uniqueSet;

std::unique\_copy(mySet.begin(), mySet.end(), std::back\_inserter(uniqueSet));

for (const auto& element : uniqueSet) {

std::cout << element << " ";

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **is\_sorted**

Пример использования алгоритма is\_sorted:

#include <iostream>

#include <algorithm>

#include <set>

int main() {

std::set<int> mySet = { 1, 2, 3, 4, 5 };

if (std::is\_sorted(mySet.begin(), mySet.end())) {

std::cout << "The set is sorted in ascending order." << std::endl;

}

else {

std::cout << "The set is not sorted." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **lexicographical\_compare**

Пример использования алгоритма lexicographical\_compare:

#include <iostream>

#include <algorithm>

#include <set>

int main() {

std::set<int> mySet1 = { 1, 2, 3, 4, 5 };

std::set<int> mySet2 = { 2, 4, 6, 8, 10 };

if (std::lexicographical\_compare(mySet1.begin(), mySet1.end(), mySet2.begin(), mySet2.end())) {

std::cout << "mySet1 is lexicographically less than mySet2." << std::endl;

}

else {

std::cout << "mySet1 is not lexicographically less than mySet2." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **merge**

Пример использования алгоритма merge:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::set<int> mySet1{ 1, 3, 5 };

std::set<int> mySet2{ 2, 4, 6 };

std::set<int> mergedSet;

std::merge(mySet1.begin(), mySet1.end(), mySet2.begin(), mySet2.end(), std::inserter(mergedSet, mergedSet.begin()));

for (const auto& element : mergedSet) {

std::cout << element << " ";

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **partial\_sort\_copy**

Пример использования алгоритма partial\_sort\_copy:

#include <iostream>

#include <set>

#include <algorithm>

#include <vector>

int main() {

std::set<int> mySet = { 5, 2, 8, 1, 9, 3 };

std::vector<int> result(3);

std::partial\_sort\_copy(mySet.begin(), mySet.end(), result.begin(), result.end());

std::cout << "Partial\_sort\_copy result: ";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **binary\_search**

Пример использования алгоритма binary\_search:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::set<int> mySet = { 5, 2, 8, 1, 6 };

int target = 8;

bool found = std::binary\_search(mySet.begin(), mySet.end(), target);

if (found) {

std::cout << "Element " << target << " found in the set." << std::endl;

}

else {

std::cout << "Element " << target << " not found in the set." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **equal\_range**

Пример использования алгоритма equal\_range:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::set<int> mySet = { 5, 2, 8, 1, 6 };

// Найдите равный диапазон для элемента

int target = 5;

auto range = mySet.equal\_range(target);

// Вывод элементов в диапазоне

std::cout << "Equal range for " << target << ": ";

for (auto it = range.first; it != range.second; ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **lower\_bound**

Пример использования алгоритма lower\_bound:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::set<int> mySet = { 5, 2, 8, 1, 6 };

// Найдите нижнюю границу для элемента

int target = 3;

auto it = mySet.lower\_bound(target);

if (it != mySet.end()) {

std::cout << "Lower bound for " << target << ": " << \*it << std::endl;

}

else {

std::cout << "No lower bound found for " << target << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **upper\_bound**

Пример использования алгоритма upper\_bound:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::set<int> mySet = { 5, 2, 8, 1, 6 };

// Найдите верхнюю границу для элемента

int target = 4;

auto it = mySet.upper\_bound(target);

if (it != mySet.end()) {

std::cout << "Upper bound for " << target << ": " << \*it << std::endl;

}

else {

std::cout << "No upper bound found for " << target << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

#### **Multiset**

##### **adjacent\_find**

##### **count**

Пример использования алгоритмаcount:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::multiset<int> multiset{ 1, 2, 2, 3, 4, 4, 5 };

int count = std::count(multiset.begin(), multiset.end(), 4);

std::cout << "Number of occurrences of 4: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **count\_if**

Пример использования алгоритмаcount\_if:

#include <iostream>

#include <set>

#include <algorithm>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

std::multiset<int> multiset{ 1, 2, 2, 3, 4, 4, 5 };

int count = std::count\_if(multiset.begin(), multiset.end(), isEven);

std::cout << "Number of even elements: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **equal**

Пример использования алгоритмаequal:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::multiset<int> multiset1{ 1, 2, 3, 4 };

std::multiset<int> multiset2{ 1, 2, 3, 4 };

bool isEqual = std::equal(multiset1.begin(), multiset1.end(), multiset2.begin(), multiset2.end());

if (isEqual) {

std::cout << "The multisets are equal" << std::endl;

}

else {

std::cout << "The multisets are not equal" << std::endl;

}

return 0;

}

Результат работы программы:

  
[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find**

Пример использования алгоритмаfind:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::multiset<int> multiset{ 1, 2, 2, 3, 4, 4, 5 };

auto it = std::find(multiset.begin(), multiset.end(), 3);

if (it != multiset.end()) {

std::cout << "Element found: " << \*it << std::endl;

}

else {

std::cout << "Element not found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find\_if**

Пример использования алгоритмаfind\_if:

#include <iostream>

#include <set>

#include <algorithm>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

std::multiset<int> multiset{ 1, 2, 2, 3, 4, 4, 5 };

auto it = std::find\_if(multiset.begin(), multiset.end(), isEven);

if (it != multiset.end()) {

std::cout << "Even element found: " << \*it << std::endl;

}

else {

std::cout << "Even element not found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find\_first\_of**

Пример использования алгоритмаfind\_first\_of:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::multiset<int> multiset{ 1, 2, 3, 4, 5 };

std::multiset<int> targets{ 2, 4, 6 };

auto it = std::find\_first\_of(multiset.begin(), multiset.end(), targets.begin(), targets.end());

if (it != multiset.end()) {

std::cout << "Element from targets found: " << \*it << std::endl;

}

else {

std::cout << "No element from targets found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **for\_each**

Пример использования алгоритмаfor\_each:

#include <iostream>

#include <set>

#include <algorithm>

void printElement(int num) {

std::cout << num << " ";

}

int main() {

std::multiset<int> multiset{ 1, 2, 3, 4, 5 };

std::for\_each(multiset.begin(), multiset.end(), printElement);

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **mismatch**

Пример использования алгоритмаmismatch:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::multiset<int> multiset1{ 1, 2, 3, 4 };

std::multiset<int> multiset2{ 1, 2, 3, 5 };

auto pair = std::mismatch(multiset1.begin(), multiset1.end(), multiset2.begin(), multiset2.end());

if (pair.first != multiset1.end()) {

std::cout << "Mismatch found: " << \*pair.first << " != " << \*pair.second << std::endl;

}

else {

std::cout << "No mismatch found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **max\_element**

Пример использования алгоритмаmax\_element:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::multiset<int> multiset{ 1, 2, 3, 4, 5 };

auto it = std::max\_element(multiset.begin(), multiset.end());

if (it != multiset.end()) {

std::cout << "Max element: " << \*it << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **min\_element**

Пример использования алгоритмаmin\_element:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::multiset<int> multiset{ 1, 2, 3, 4, 5 };

auto it = std::min\_element(multiset.begin(), multiset.end());

if (it != multiset.end()) {

std::cout << "Min element: " << \*it << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **accumulate**

Пример использования алгоритмаaccumulate:

#include <iostream>

#include <set>

#include <algorithm>

#include <numeric>

int main() {

std::multiset<int> mySet{ 1, 2, 3, 4, 5 };

int sum = std::accumulate(mySet.begin(), mySet.end(), 0);

std::cout << "Sum of elements: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **adjacent\_difference**

Пример использования алгоритмаadjacent\_difference**:**

#include <iostream>

#include <set>

#include <algorithm>

#include <numeric>

int main() {

std::multiset<int> mySet{ 1, 2, 3, 4, 5 };

std::multiset<int> result;

std::adjacent\_difference(mySet.begin(), mySet.end(), std::inserter(result, result.begin()));

for (int num: result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **inner\_product**

Пример использования алгоритмаinner\_product**:**

#include <iostream>

#include <set>

#include <algorithm>

#include <numeric>

int main() {

std::multiset<int> set1{ 1, 2, 3 };

std::multiset<int> set2{ 4, 5, 6 };

int result = std::inner\_product(set1.begin(), set1.end(), set2.begin(), 0);

std::cout << "Inner product: " << result << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **partial\_sum**

Пример использования алгоритмаpartial\_sum**:**

#include <iostream>

#include <set>

#include <algorithm>

#include <numeric>

int main() {

std::multiset<int> mySet{ 1, 2, 3, 4, 5 };

std::multiset<int> result;

std::partial\_sum(mySet.begin(), mySet.end(), std::inserter(result, result.begin()));

for (int num: result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **copy**

Пример использования алгоритмаcopy:

#include <iostream>

#include <set>

#include <vector>

#include <algorithm>

int main() {

std::multiset<int> ms = { 1, 2, 3, 4, 5 };

std::cout << "Source multiset:";

for (const auto& value: ms) {

std::cout << " " << value;

}

std::cout << std::endl;

std::vector<int> vec;

std::copy(ms.begin(), ms.end(), std::back\_inserter(vec));

std::cout << "Copied vector:";

for (const auto& value: vec) {

std::cout << " " << value;

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <set>

#include <algorithm>

int square(int x) {

return x \* x;

}

int main() {

std::multiset<int> inputSet = { 1, 2, 3, 4, 5 };

std::multiset<int> outputSet;

std::transform(inputSet.begin(), inputSet.end(), std::inserter(outputSet, outputSet.begin()), square);

std::cout << "Input Set: ";

for (const auto& elem: inputSet) {

std::cout << elem << " ";

}

std::cout << std::endl;

std::cout << "Output Set: ";

for (const auto& elem: outputSet) {

std::cout << elem << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **lexicographical\_compare**

Пример использования алгоритмаlexicographical\_compare:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::multiset<int> set1 = { 1, 2, 3 };

std::multiset<int> set2 = { 1, 2, 4 };

bool isSet1LessThanSet2 = std::lexicographical\_compare(set1.begin(), set1.end(), set2.begin(), set2.end());

if (isSet1LessThanSet2) {

std::cout << "Set1 is less than Set2" << std::endl;

}

else {

std::cout << "Set1 is not less than Set2" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **merge**

Пример использования алгоритмаmerge:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::multiset<int> set1 = { 1, 3, 5 };

std::multiset<int> set2 = { 2, 4, 6 };

std::multiset<int> mergedSet;

std::merge(set1.begin(), set1.end(), set2.begin(), set2.end(), std::inserter(mergedSet, mergedSet.begin()));

std::cout << "Merged Set: ";

for (const auto& elem: mergedSet) {

std::cout << elem << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **lower\_bound**

Пример использования алгоритмаlower\_bound:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::multiset<int> set = { 2, 4, 6, 8, 10 };

int value = 5;

auto it = set.lower\_bound(value);

if (it != set.end()) {

std::cout << "First element not less than " << value << ": " << \*it << std::endl;

}

else {

std::cout << "No element not less than " << value << " found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **upper\_bound**

Пример использования алгоритмаupper\_bound:

#include <iostream>

#include <set>

#include <algorithm>

int main() {

std::multiset<int> set = { 2, 4, 6, 8, 10 };

int value = 5;

auto it = set.upper\_bound(value);

if (it != set.end()) {

std::cout << "First element greater than " << value << ": " << \*it << std::endl;

}

else {

std::cout << "No element greater than " << value << " found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

## **Map**

#### **Map**

##### **adjacent\_find**

Пример использования алгоритма adjacent\_find:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::map<int, std::string> myMap = { {1, "one"}, {2, "two"}, {3, "two"}, {4, "three"}, {5, "four"} };

auto it = std::adjacent\_find(myMap.begin(), myMap.end(),

[](const auto& a, const auto& b) {

return a.second == b.second;

});

if (it != myMap.end()) {

std::cout << "Adjacent elements with the same value found: "

<< it->first << " and " << std::next(it)->first << std::endl;

}

else {

std::cout << "No adjacent elements with the same value found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **count\_if**

Пример использования алгоритма count\_if:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::map<int, int> myMap = { {1, 10}, {2, 20}, {3, 30}, {4, 20}, {5, 10} };

int count = std::count\_if(myMap.begin(), myMap.end(),

[](const auto& pair) {

return pair.second > 15;

});

std::cout << "Count of elements with value greater than 15: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **equal**

Пример использования алгоритма equal:

#include <iostream>

#include <map>

int main() {

setlocale(LC\_ALL, "Russian");

std::map<int, int> map1{ {1, 10}, {2, 20}, {3, 30} };

std::map<int, int> map2{ {1, 10}, {2, 20}, {3, 30} };

bool isEqual = std::equal(map1.begin(), map1.end(), map2.begin());

std::cout << "map1 и map2 " << (isEqual ? "равны" : "не равны") << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find**

Пример использования алгоритма find:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::map<int, int> myMap;

// Заполняем карту значениями

myMap[1] = 10;

myMap[2] = 20;

myMap[3] = 10;

myMap[4] = 30;

myMap[5] = 10;

int keyToFind = 3; // Ключ, который мы хотим найти

// Используем алгоритм std::find

auto it = std::find\_if(myMap.begin(), myMap.end(), [keyToFind](const std::pair<int, int>& element) {

return element.first == keyToFind;

});

if (it != myMap.end()) {

std::cout << "Элемент найден! Значение: " << it->second << std::endl;

}

else {

std::cout << "Элемент не найден." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find\_if**

Пример использования алгоритма find\_if:

#include <iostream>

#include <map>

#include <algorithm>

bool isEven(const std::pair<int, int>& element) {

return element.second % 2 == 0;

}

int main() {

std::map<int, int> myMap = { {1, 10}, {2, 20}, {3, 30}, {4, 40}, {5, 50} };

auto it = std::find\_if(myMap.begin(), myMap.end(), isEven);

if (it != myMap.end()) {

std::cout << "Element found: " << it->first << " - " << it->second << std::endl;

}

else {

std::cout << "Element not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find\_end**

Пример использования алгоритма find\_end:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::map<int, std::string> myMap = { {1, "one"}, {2, "two"}, {3, "three"}, {4, "four"}, {5, "five"} };

std::map<int, std::string> subMap = { {2, "two"}, {3, "three"} };

auto it = std::find\_end(myMap.begin(), myMap.end(), subMap.begin(), subMap.end());

if (it != myMap.end()) {

std::cout << "Subsequence found starting at key: " << it->first << std::endl;

}

else {

std::cout << "Subsequence not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find\_first\_of**

Пример использования алгоритма find\_first\_of:

#include <iostream>

#include <map>

#include <algorithm>

#include <vector>

int main() {

setlocale(LC\_ALL, "Russian");

std::map<int, int> myMap;

// Заполняем карту значениями

myMap[1] = 10;

myMap[2] = 20;

myMap[3] = 10;

myMap[4] = 30;

myMap[5] = 10;

std::vector<int> valuesToFind = { 20, 30 }; // Значения, которые мы хотим найти

// Используем алгоритм std::find\_first\_of

auto it = std::find\_first\_of(myMap.begin(), myMap.end(), valuesToFind.begin(), valuesToFind.end(),

[](const std::pair<int, int>& element, int value) {

return element.second == value;

});

if (it != myMap.end()) {

std::cout << "Первое совпадение найдено! Ключ: " << it->first << ", Значение: " << it->second << std::endl;

}

else {

std::cout << "Совпадение не найдено." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **for\_each**

Пример использования алгоритма for\_each:

#include <iostream>

#include <map>

#include <algorithm>

void printPair(const std::pair<int, std::string>& pair) {

std::cout << pair.first << " - " << pair.second << std::endl;

}

int main() {

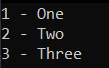
std::map<int, std::string> map = { {1, "One"}, {2, "Two"}, {3, "Three"} };

std::for\_each(map.begin(), map.end(), printPair);

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **mismatch**

Пример использования алгоритма mismatch:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::map<int, int> map1;

std::map<int, int> map2;

// Заполняем первую карту значениями

map1[1] = 10;

map1[2] = 20;

map1[3] = 30;

// Заполняем вторую карту значениями

map2[1] = 10;

map2[2] = 25; // Измененное значение

map2[3] = 30;

// Используем алгоритм std::mismatch

auto it = std::mismatch(map1.begin(), map1.end(), map2.begin());

if (it.first == map1.end() && it.second == map2.end()) {

std::cout << "Карты идентичны." << std::endl;

}

else {

std::cout << "Карты отличаются на позиции: " << it.first->first << std::endl;

std::cout << "Значение в первой карте: " << it.first->second << std::endl;

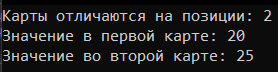
std::cout << "Значение во второй карте: " << it.second->second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **search**

Пример использования алгоритма search:

#include <iostream>

#include <map>

#include <algorithm>

#include <vector>

int main() {

std::map<int, std::string> myMap = { {1, "Apple"}, {2, "Banana"}, {3, "Orange"}, {4, "Mango"} };

std::vector<std::string> sequence = { "Banana", "Orange" };

auto it = std::search(myMap.begin(), myMap.end(), sequence.begin(), sequence.end(),

[](const auto& mapElement, const auto& sequenceElement) {

return mapElement.second == sequenceElement;

});

if (it != myMap.end()) {

std::cout << "Sequence found starting at key " << it->first << std::endl;

}

else {

std::cout << "Sequence not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **search\_n**

Пример использования алгоритма search\_n:

#include <iostream>

#include <map>

#include <algorithm>

#include <vector>

int main() {

std::map<int, std::string> myMap = { {1, "Apple"}, {2, "Banana"}, {3, "Orange"}, {4, "Mango"} };

std::vector<std::string> sequence = { "Banana", "Orange" };

auto it = std::search(myMap.begin(), myMap.end(), sequence.begin(), sequence.end(),

[](const auto& mapElement, const auto& sequenceElement) {

return mapElement.second == sequenceElement;

});

if (it != myMap.end()) {

std::cout << "Sequence found starting at key " << it->first << std::endl;

}

else {

std::cout << "Sequence not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **max\_element**

Пример использования алгоритма max\_element:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::map<int, int> myMap = { {1, 10}, {2, 20}, {3, 30}, {4, 40}, {5, 50} };

auto maxElement = std::max\_element(myMap.begin(), myMap.end(),

[](const auto& lhs, const auto& rhs) {

return lhs.second < rhs.second;

});

if (maxElement != myMap.end()) {

std::cout << "Maximum element: Key = " << maxElement->first << ", Value = " << maxElement->second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **min\_element**

Пример использования алгоритма min\_element:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::map<int, int> myMap = { {1, 10}, {2, 20}, {3, 30}, {4, 40}, {5, 5} };

auto minElement = std::min\_element(myMap.begin(), myMap.end(),

[](const auto& lhs, const auto& rhs) {

return lhs.second < rhs.second;

});

if (minElement != myMap.end()) {

std::cout << "Minimum element: Key = " << minElement->first << ", Value = " << minElement->second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **accumulate**

Пример использования алгоритма accumulate:

#include <iostream>

#include <map>

#include <numeric>

int main() {

std::map<int, int> myMap = { {1, 10}, {2, 20}, {3, 30}, {4, 40}, {5, 50} };

int sum = std::accumulate(myMap.begin(), myMap.end(), 0,

[](int currentSum, const auto& mapElement) {

return currentSum + mapElement.second;

});

std::cout << "Sum of all values: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **swap**

Пример использования алгоритма swap:

#include <iostream>

#include <map>

int main() {

setlocale(LC\_ALL, "Russian");

std::map<int, int> myMap1 = { {1, 10}, {2, 20} };

std::map<int, int> myMap2 = { {3, 30}, {4, 40} };

std::cout << "До swap:\n";

std::cout << "myMap1: ";

for (const auto& pair : myMap1) {

std::cout << pair.first << ":" << pair.second << " ";

}

std::cout << "\nmyMap2: ";

for (const auto& pair : myMap2) {

std::cout << pair.first << ":" << pair.second << " ";

}

std::cout << std::endl;

std::swap(myMap1, myMap2);

std::cout << "После swap:\n";

std::cout << "myMap1: ";

for (const auto& pair : myMap1) {

std::cout << pair.first << ":" << pair.second << " ";

}

std::cout << "\nmyMap2: ";

for (const auto& pair : myMap2) {

std::cout << pair.first << ":" << pair.second << " ";

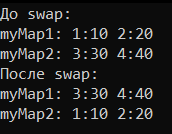
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **replace\_copy\_if**

Пример использования алгоритма replace\_copy\_if:

#include <iostream>

#include <map>

#include <algorithm>

#include <iterator>

bool isOdd(const std::pair<int, int>& pair) {

return pair.second % 2 != 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

std::map<int, int> myMap = { {1, 10}, {2, 20}, {3, 30} };

std::map<int, int> result;

std::replace\_copy\_if(myMap.begin(), myMap.end(), std::inserter(result, result.begin()),

isOdd, std::make\_pair(0, 0));

std::cout << "Результат: ";

for (const auto& pair : result) {

std::cout << pair.first << ":" << pair.second << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **remove\_copy\_if**

Пример использования алгоритма replace\_copy\_if:

#include <iostream>

#include <map>

#include <algorithm>

#include <vector>

int main() {

std::map<int, int> myMap = { {1, 10}, {2, 20}, {3, 30} };

std::vector<std::pair<int, int>> result;

// Пример использования remove\_copy\_if для копирования элементов из std::map

// исключая элементы с ключом 2

std::remove\_copy\_if(myMap.begin(), myMap.end(), std::back\_inserter(result),

[](const std::pair<int, int>& pair) {

return pair.first == 2;

});

// Вывод результата

for (const auto& pair : result) {

std::cout << pair.first << ": " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **copy**

Пример использования алгоритма copy:

#include <iostream>

#include <map>

#include <algorithm>

#include <vector>

int main() {

std::map<int, int> myMap = { {1, 10}, {2, 20}, {3, 30} };

std::vector<std::pair<int, int>> result(myMap.size());

// Пример использования copy для копирования элементов из std::map в std::vector

std::copy(myMap.begin(), myMap.end(), result.begin());

// Вывод результата

for (const auto& pair : result) {

std::cout << pair.first << ": " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <map>

#include <algorithm>

#include <iterator>

int main() {

std::map<int, int> myMap{ {1, 10}, {2, 20}, {3, 30}, {4, 40}, {5, 50} };

std::map<int, int> transformedMap;

std::transform(myMap.begin(), myMap.end(), std::inserter(transformedMap, transformedMap.begin()),

[](const std::pair<int, int>& pair) {

return std::make\_pair(pair.first, pair.second \* 2);

});

std::cout << "Original Map:" << std::endl;

for (const auto& pair : myMap) {

std::cout << pair.first << ": " << pair.second << std::endl;

}

std::cout << "Transformed Map:" << std::endl;

for (const auto& pair : transformedMap) {

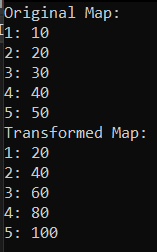
std::cout << pair.first << ": " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **unique\_copy**

Пример использования алгоритма unique\_copy:

#include <iostream>

#include <map>

#include <algorithm>

#include <iterator>

int main() {

std::map<int, int> myMap{ {1, 10}, {2, 20}, {3, 20}, {4, 30}, {5, 30} };

std::map<int, int> uniqueCopyMap;

std::unique\_copy(myMap.begin(), myMap.end(), std::inserter(uniqueCopyMap, uniqueCopyMap.begin()));

std::cout << "Original Map:" << std::endl;

for (const auto& pair : myMap) {

std::cout << pair.first << ": " << pair.second << std::endl;

}

std::cout << "Unique Copy Map:" << std::endl;

for (const auto& pair : uniqueCopyMap) {

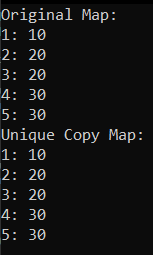
std::cout << pair.first << ": " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **is\_sorted**

Пример использования алгоритма is\_sorted:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::map<int, std::string> sortedMap{ {1, "one"}, {2, "two"}, {3, "three"}, {4, "four"}, {5, "five"} };

std::map<int, std::string> unsortedMap{ {1, "one"}, {5, "five"}, {3, "three"}, {2, "two"}, {4, "four"} };

bool isSorted1 = std::is\_sorted(sortedMap.begin(), sortedMap.end());

bool isSorted2 = std::is\_sorted(unsortedMap.begin(), unsortedMap.end());

std::cout << "Is sortedMap sorted? " << std::boolalpha << isSorted1 << std::endl;

std::cout << "Is unsortedMap sorted? " << std::boolalpha << isSorted2 << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **lexicographical\_compare**

Пример использования алгоритма lexicographical\_compare:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::map<int, std::string> map1{ {1, "one"}, {2, "two"}, {3, "three"} };

std::map<int, std::string> map2{ {1, "one"}, {2, "two"}, {4, "four"} };

bool isMap1LessThanMap2 = std::lexicographical\_compare(map1.begin(), map1.end(), map2.begin(), map2.end());

std::cout << "Is map1 lexicographically less than map2? " << std::boolalpha << isMap1LessThanMap2 << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **merge**

Пример использования алгоритма merge:

#include <iostream>

#include <map>

#include <algorithm>

#include <iterator>

int main() {

std::map<int, std::string> map1{ {1, "one"}, {3, "three"}, {5, "five"} };

std::map<int, std::string> map2{ {2, "two"}, {4, "four"}, {6, "six"} };

std::map<int, std::string> mergedMap;

std::merge(map1.begin(), map1.end(), map2.begin(), map2.end(), std::inserter(mergedMap, mergedMap.begin()));

std::cout << "Merged Map:" << std::endl;

for (const auto& pair : mergedMap) {

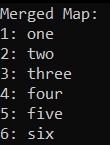
std::cout << pair.first << ": " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **partial\_sort\_copy**

Пример использования алгоритма partial\_sort\_copy:

#include <iostream>

#include <map>

#include <algorithm>

#include <vector>

int main() {

std::map<int, std::string> myMap{ {3, "three"}, {1, "one"}, {5, "five"}, {2, "two"}, {4, "four"} };

std::vector<std::pair<int, std::string>> partialSorted;

std::partial\_sort\_copy(myMap.begin(), myMap.end(), partialSorted.begin(), partialSorted.end(),

[](const std::pair<int, std::string>& pair1, const std::pair<int, std::string>& pair2) {

return pair1.first < pair2.first;

});

std::cout << "Original Map:" << std::endl;

for (const auto& pair : myMap) {

std::cout << pair.first << ": " << pair.second << std::endl;

}

std::cout << "Partial Sorted Map:" << std::endl;

for (const auto& pair : partialSorted) {

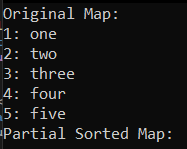
std::cout << pair.first << ": " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **equal\_range**

Пример использования алгоритма equal\_range:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::map<int, std::string> myMap{ {1, "one"}, {2, "two"}, {2, "duplicate"}, {3, "three"}, {4, "four"}, {5, "five"} };

auto range = myMap.equal\_range(2);

std::cout << "Elements with key 2:" << std::endl;

for (auto it = range.first; it != range.second; ++it) {

std::cout << it->first << ": " << it->second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **lower\_bound**

Пример использования алгоритма lower\_bound:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::map<int, std::string> myMap{ {1, "one"}, {2, "two"}, {3, "three"}, {4, "four"}, {5, "five"} };

int keyToSearch = 3;

auto it = myMap.lower\_bound(keyToSearch);

if (it != myMap.end()) {

std::cout << "Element with key " << keyToSearch << " found: " << it->first << ": " << it->second << std::endl;

}

else {

std::cout << "Element with key " << keyToSearch << " not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **upper\_bound**

Пример использования алгоритма upper\_bound:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::map<int, std::string> myMap{ {1, "one"}, {2, "two"}, {3, "three"}, {4, "four"}, {5, "five"} };

int keyToSearch = 3;

auto it = myMap.upper\_bound(keyToSearch);

if (it != myMap.end()) {

std::cout << "Element with key " << keyToSearch << " found: " << it->first << ": " << it->second << std::endl;

}

else {

std::cout << "Element with key " << keyToSearch << " not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

#### **Multimap**

##### **find\_if**

Пример использования алгоритмаfind\_if:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::multimap<int, std::string> myMap = { {1, "apple"}, {2, "banana"}, {3, "cherry"} };

auto it = std::find\_if(myMap.begin(), myMap.end(), [](const auto& pair) {

return pair.first == 2; // Ищем элемент с ключом 2

});

if (it != myMap.end()) {

std::cout << "Element found: " << it->second << std::endl;

}

else {

std::cout << "Element not found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **find\_end**

Пример использования алгоритмаfind\_end:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::multimap<int, std::string> myMap = { {1, "apple"}, {2, "banana"}, {3, "cherry"},

{4, "banana"}, {5, "apple"}, {6, "banana"} };

std::multimap<int, std::string> sequence = { {2, "banana"}, {3, "cherry"} };

auto it = std::find\_end(myMap.begin(), myMap.end(), sequence.begin(), sequence.end());

if (it != myMap.end()) {

std::cout << "Last occurrence found at key: " << it->first << std::endl;

}

else {

std::cout << "Sequence not found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **for\_each**

Пример использования алгоритмаfor\_each:

#include <iostream>

#include <map>

#include <algorithm>

void PrintPair(const std::pair<const int, std::string>& pair) {

std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

int main() {

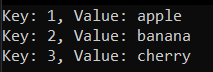
std::multimap<int, std::string> myMap = { {1, "apple"}, {2, "banana"}, {3, "cherry"} };

std::for\_each(myMap.begin(), myMap.end(), PrintPair);

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **mismatch**

Пример использования алгоритмаmismatch:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::multimap<int, std::string> map1 = { {1, "apple"}, {2, "banana"}, {3, "cherry"} };

std::multimap<int, std::string> map2 = { {1, "apple"}, {2, "kiwi"}, {3, "cherry"} };

auto mismatchIt = std::mismatch(map1.begin(), map1.end(), map2.begin());

if (mismatchIt.first != map1.end()) {

std::cout << "First mismatch: Key=" << mismatchIt.first->first

<< ", Value1=" << mismatchIt.first->second

<< ", Value2=" << mismatchIt.second->second << std::endl;

}

else {

std::cout << "No mismatch" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **search**

Пример использования алгоритмаsearch:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::multimap<int, std::string> myMap = { {1, "apple"}, {2, "banana"}, {3, "cherry"},

{4, "banana"}, {5, "apple"}, {6, "banana"} };

std::multimap<int, std::string> sequence = { {2, "banana"}, {3, "cherry"} };

auto it = std::search(myMap.begin(), myMap.end(), sequence.begin(), sequence.end());

if (it != myMap.end()) {

std::cout << "First occurrence found at key: " << it->first << std::endl;

}

else {

std::cout << "Sequence not found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **max\_element**

Пример использования алгоритмаmax\_element:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::multimap<int, std::string> myMap = { {1, "apple"}, {2, "banana"}, {3, "cherry"} };

auto maxIt = std::max\_element(myMap.begin(), myMap.end());

if (maxIt != myMap.end()) {

std::cout << "Max element: Key=" << maxIt->first << ", Value=" << maxIt->second << std::endl;

}

else {

std::cout << "Empty multimap" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **min\_element**

Пример использования алгоритмаmin\_element:

##include <iostream>

#include <map>

#include <algorithm>

int main() {

std::multimap<int, std::string> myMap = { {1, "apple"}, {2, "banana"}, {3, "cherry"} };

auto minIt = std::min\_element(myMap.begin(), myMap.end());

if (minIt != myMap.end()) {

std::cout << "Min element: Key=" << minIt->first << ", Value=" << minIt->second << std::endl;

}

else {

std::cout << "Empty multimap" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **accumulate**

Пример использования алгоритмаaccumulate:

#include <iostream>

#include <map>

#include <numeric>

int main() {

std::multimap<int, int> myMap = { {1, 10}, {2, 20}, {3, 30} };

int sum = std::accumulate(myMap.begin(), myMap.end(), 0,

[](int currentSum, const auto& pair) {

return currentSum + pair.second;

});

std::cout << "Sum of values: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **swap**

Пример использования алгоритмаswap**:**

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::multimap<int, int> myMap = { {1, 10}, {2, 20}, {3, 30} };

auto it1 = myMap.find(1);

auto it2 = myMap.find(3);

if (it1 != myMap.end() && it2 != myMap.end()) {

std::swap(it1->second, it2->second);

}

// Выводим содержимое multimap

for (const auto& pair : myMap) {

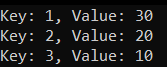
std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **copy**

Пример использования алгоритмаcopy:

#include <iostream>

#include <map>

#include <algorithm>

#include <iterator>

int main() {

std::multimap<int, int> myMap = { {1, 10}, {2, 20}, {3, 30}, {4, 40} };

std::map<int, int> resultMap;

std::copy(myMap.begin(), myMap.end(), std::inserter(resultMap, resultMap.begin()));

// Выводим содержимое результирующего map

for (const auto& pair : resultMap) {

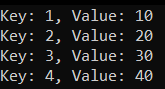
std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **reverse\_copy**

Пример использования алгоритмаreverse\_copy:

#include <iostream>

#include <map>

#include <algorithm>

#include <vector>

int main() {

std::multimap<int, int> myMap = { {1, 10}, {2, 20}, {3, 30}, {4, 40} };

std::vector<std::pair<int, int>> result;

std::reverse\_copy(myMap.begin(), myMap.end(), std::back\_inserter(result));

// Выводим содержимое результирующего вектора

for (const auto& pair : result) {

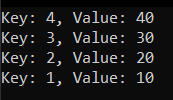
std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <map>

#include <algorithm>

#include <vector>

int main() {

std::multimap<int, int> myMap = { {1, 10}, {2, 20}, {3, 30}, {4, 40} };

std::vector<int> values = { 1, 2, 3, 4 };

std::transform(myMap.begin(), myMap.end(), values.begin(), [](const auto& pair) {

return pair.second \* 2;

});

// Выводим обновленные значения

for (const auto& value : values) {

std::cout << "Value: " << value << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **unique\_copy**

Пример использования алгоритма unique\_copy:

#include <iostream>

#include <map>

#include <algorithm>

#include <vector>

int main() {

std::multimap<int, int> myMap = { {1, 10}, {1, 20}, {2, 30}, {3, 40} };

std::vector<std::pair<int, int>> result;

std::unique\_copy(myMap.begin(), myMap.end(), std::back\_inserter(result));

// Выводим содержимое результирующего вектора

for (const auto& pair : result) {

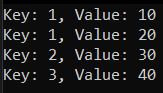
std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **is\_sorted**

Пример использования алгоритма is\_sorted:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::multimap<int, int> myMap1 = {{1, 10}, {2, 20}, {3, 30}, {4, 40}};

std::multimap<int, int> myMap2 = {{1, 10}, {3, 30}, {2, 20}, {4, 40}};

bool sorted1 = std::is\_sorted(myMap1.begin(), myMap1.end());

bool sorted2 = std::is\_sorted(myMap2.begin(), myMap2.end());

std::cout << "myMap1 is " << (sorted1 ? "sorted" : "not sorted") << std::endl;

std::cout << "myMap2 is " << (sorted2 ? "sorted" : "not sorted") << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **lexicographical\_compare**

Пример использования алгоритмаlexicographical\_compare:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::multimap<int, int> myMap1 = { {1, 10}, {2, 20}, {3, 30} };

std::multimap<int, int> myMap2 = { {1, 10}, {2, 20}, {4, 40} };

bool less = std::lexicographical\_compare(myMap1.begin(), myMap1.end(), myMap2.begin(), myMap2.end());

std::cout << "myMap1 is " << (less ? "less than" : "greater than or equal to") << " myMap2 lexicographically" << std::endl;

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **merge**

Пример использования алгоритмаmerge:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::multimap<int, int> myMap1 = { {1, 10}, {3, 30} };

std::multimap<int, int> myMap2 = { {2, 20}, {4, 40} };

std::multimap<int, int> mergedMap;

std::merge(myMap1.begin(), myMap1.end(), myMap2.begin(), myMap2.end(), std::inserter(mergedMap, mergedMap.begin()));

// Выводим содержимое объединенного multimap

for (const auto& pair : mergedMap) {

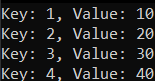
std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

##### **lower\_bound**

Пример использования алгоритмаlower\_bound:

#include <iostream>

#include <map>

#include <algorithm>

int main() {

std::multimap<int, int> myMap = { {1, 10}, {2, 20}, {2, 30}, {3, 40}, {3, 50} };

int key = 2;

auto it = std::lower\_bound(myMap.begin(), myMap.end(), key, [](const auto& pair1, const auto& pair2) {

return pair1.first < pair2;

});

if (it != myMap.end()) {

std::cout << "Key: " << it->first << ", Value: " << it->second << std::endl;

}

else {

std::cout << "Key not found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Ассоциативные контейнеры>](#_Ассоциативные_контейнеры_2)

# **Неупорядоченные ассоциативные контейнеры**

## **Unordered\_set**

#### **Unordered\_set**

##### **adjacent\_find**

Пример использования алгоритмаadjacent\_find:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

setlocale(LC\_ALL, "Russian");

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5, 5, 6, 7 };

auto it = std::adjacent\_find(mySet.begin(), mySet.end());

if (it != mySet.end()) {

std::cout << "Найдены соседние элементы с одинаковыми значениями: " << \*it << std::endl;

}

else {

std::cout << "Соседние элементы с одинаковыми значениями не найдены." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **count**

Пример использования алгоритмаcount:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

int count = std::count(mySet.begin(), mySet.end(), 3);

std::cout << "Count of element 3: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **count\_if**

Пример использования алгоритмаcount\_if:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

int count = std::count\_if(mySet.begin(), mySet.end(), isEven);

std::cout << "Count of even elements: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **equal**

Пример использования алгоритмаequal:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> set1 = { 1, 2, 3, 4, 5 };

std::unordered\_set<int> set2 = { 1, 2, 3, 4, 5 };

bool isEqual = std::equal(set1.begin(), set1.end(), set2.begin(), set2.end());

if (isEqual) {

std::cout << "Sets are equal." << std::endl;

}

else {

std::cout << "Sets are not equal." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find**

Пример использования алгоритмаfind:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

auto it = std::find(mySet.begin(), mySet.end(), 3);

if (it != mySet.end()) {

std::cout << "Element found: " << \*it << std::endl;

}

else {

std::cout << "Element not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find\_if**

Пример использования алгоритмаfind\_if:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

auto it = std::find\_if(mySet.begin(), mySet.end(), isEven);

if (it != mySet.end()) {

std::cout << "First even element found: " << \*it << std::endl;

}

else {

std::cout << "No even elements found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find\_end**

Пример использования алгоритмаfind\_end:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <vector>

int main() {

std::unordered\_set<int> set = { 1, 2, 3, 4, 5, 1, 2, 3, 4, 5 };

std::vector<int> subsequence = { 1, 2, 3 };

auto it = std::find\_end(set.begin(), set.end(), subsequence.begin(), subsequence.end());

if (it != set.end()) {

std::cout << "Subsequence found at position: " << std::distance(set.begin(), it) << std::endl;

}

else {

std::cout << "Subsequence not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find\_first\_of**

Пример использования алгоритмаfind\_first\_of:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

**#i**nclude <vector>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5, 6, 7, 8 };

std::vector<int> values = { 7, 8, 9 };

auto it = std::find\_first\_of(mySet.begin(), mySet.end(), values.begin(), values.end());

if (it != mySet.end()) {

std::cout << "Value found: " << \*it << std::endl;

}

else {

std::cout << "No matching values found." << std::endl;

}

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **for\_each**

Пример использования алгоритмаfor\_each:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

void printElement(int num) {

std::cout << num << " ";

}

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

std::for\_each(mySet.begin(), mySet.end(), printElement);

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **mismatch**

Пример использования алгоритмаmismatch:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <vector>

int main() {

std::unordered\_set<int> set1 = { 1, 2, 3, 4, 5 };

std::vector<int> set2 = { 1, 2, 3, 5, 6 };

auto pair = std::mismatch(set1.begin(), set1.end(), set2.begin());

if (pair.first != set1.end() && pair.second != set2.end()) {

std::cout << "Mismatch found: " << \*pair.first << " and " << \*pair.second << std::endl;

}

else {

std::cout << "No mismatch found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **search**

Пример использования алгоритмаsearch:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <vector>

int main() {

std::unordered\_set<int> set = { 1, 2, 3, 4, 5 };

std::vector<int> subsequence = { 3, 4 };

auto it = std::search(set.begin(), set.end(), subsequence.begin(), subsequence.end());

if (it != set.end()) {

std::cout << "Subsequence found at position: " << std::distance(set.begin(), it) << std::endl;

}

else {

std::cout << "Subsequence not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **search\_n**

Пример использования алгоритмаsearch\_n:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> set = { 1, 2, 2, 2, 3, 4, 5 };

int count = 3;

int value = 2;

auto it = std::search\_n(set.begin(), set.end(), count, value);

if (it != set.end()) {

std::cout << "Consecutive " << count << " occurrences of " << value << " found." << std::endl;

}

else {

std::cout << "Consecutive " << count << " occurrences of " << value << " not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **max\_element**

Пример использования алгоритмаmax\_element:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> mySet = { 3, 1, 4, 1, 5, 9, 2, 6 };

auto it = std::max\_element(mySet.begin(), mySet.end());

if (it != mySet.end()) {

std::cout << "Max element: " << \*it << std::endl;

}

else {

std::cout << "Set is empty." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **min\_element**

Пример использования алгоритмаmin\_element:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> mySet = { 3, 1, 4, 1, 5, 9, 2, 6 };

auto it = std::min\_element(mySet.begin(), mySet.end());

if (it != mySet.end()) {

std::cout << "Min element: " << \*it << std::endl;

}

else {

std::cout << "Set is empty." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **accumulate**

Пример использования алгоритмаaccumulate:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <numeric>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

int sum = std::accumulate(mySet.begin(), mySet.end(), 0);

std::cout << "Sum of elements: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **adjacent\_difference**

Пример использования алгоритмаadjacent\_difference:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <numeric>

#include <vector>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

std::vector<int> differences(mySet.size());

std::adjacent\_difference(mySet.begin(), mySet.end(), differences.begin());

std::cout << "Differences between adjacent elements: ";

for (int diff : differences) {

std::cout << diff << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **inner\_product**

Пример использования алгоритмаinner\_product:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <numeric>

#include <vector>

int main() {

std::unordered\_set<int> set1 = { 1, 2, 3, 4, 5 };

std::unordered\_set<int> set2 = { 2, 4, 6, 8, 10 };

int result = std::inner\_product(set1.begin(), set1.end(), set2.begin(), 0);

std::cout << "Inner product: " << result << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **partial\_sum**

Пример использования алгоритмаpartial\_sum:

#include <iostream>

#include <unordered\_set>

#include <numeric>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

std::cout << "Original set: ";

for (const auto& element : mySet) {

std::cout << element << " ";

}

std::cout << std::endl;

std::unordered\_set<int> result;

std::partial\_sum(mySet.begin(), mySet.end(), std::inserter(result, result.begin()));

std::cout << "Partial sum set: ";

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **swap**

Пример использования алгоритмаswap:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> set1 = { 1, 2, 3 };

std::unordered\_set<int> set2 = { 4, 5, 6 };

std::cout << "Before swap:\n";

std::cout << "Set 1: ";

for (const auto& element : set1) {

std::cout << element << " ";

}

std::cout << "\nSet 2: ";

for (const auto& element : set2) {

std::cout << element << " ";

}

std::cout << std::endl;

std::swap(set1, set2);

std::cout << "After swap:\n";

std::cout << "Set 1: ";

for (const auto& element : set1) {

std::cout << element << " ";

}

std::cout << "\nSet 2: ";

for (const auto& element : set2) {

std::cout << element << " ";

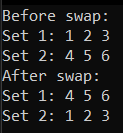
}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **replace\_copy**

Пример использования алгоритмаreplace\_copy:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

#include <vector>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

std::vector<int> result;

std::replace\_copy(mySet.begin(), mySet.end(), std::back\_inserter(result), 3, 6);

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **replace\_copy\_if**

Пример использования алгоритмаreplace\_copy\_if:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

#include <vector>

bool isOdd(int number) {

return number % 2 != 0;

}

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

std::vector<int> result;

std::replace\_copy\_if(mySet.begin(), mySet.end(), std::back\_inserter(result), isOdd, 0);

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **remove\_copy**

Пример использования алгоритмаremove\_copy:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

#include <vector>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

std::vector<int> result;

std::remove\_copy(mySet.begin(), mySet.end(), std::back\_inserter(result), 3);

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



##### **remove\_copy\_if**

Пример использования алгоритмаremove\_copy\_if:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

#include <vector>

bool isOdd(int number) {

return number % 2 != 0;

}

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

std::vector<int> result;

std::remove\_copy\_if(mySet.begin(), mySet.end(), std::back\_inserter(result), isOdd);

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



##### **copy**

Пример использования алгоритма copy:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <vector>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

std::vector<int> myVector(5);

std::copy(mySet.begin(), mySet.end(), myVector.begin());

for (const auto& element : myVector) {

std::cout << element << " ";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **copy\_backward**

Пример использования алгоритма copy\_backward:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

#include <vector>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

std::vector<int> result(mySet.size());

std::copy\_backward(mySet.begin(), mySet.end(), result.end());

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **reverse\_copy**

Пример использования алгоритма reverse\_copy:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

#include <vector>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

std::vector<int> result;

std::reverse\_copy(mySet.begin(), mySet.end(), std::back\_inserter(result));

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <vector>

#include <cmath>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

std::vector<double> myVector(5);

std::transform(mySet.begin(), mySet.end(), myVector.begin(), [](int value) {

return std::sqrt(value);

});

for (const auto& element : myVector) {

std::cout << element << " ";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **unique\_copy**

Пример использования алгоритма unique:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

#include <vector>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 2, 3, 4, 4, 5 };

std::vector<int> result;

std::unique\_copy(mySet.begin(), mySet.end(), std::back\_inserter(result));

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **is\_sorted**

Пример использования алгоритма is\_sorted:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 3, 4, 5 };

if (std::is\_sorted(mySet.begin(), mySet.end())) {

std::cout << "The set is sorted" << std::endl;

}

else {

std::cout << "The set is not sorted" << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **lexicographical\_compare**

Пример использования алгоритма lexicographical\_compare:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> set1 = { 1, 2, 3 };

std::unordered\_set<int> set2 = { 1, 2, 4 };

if (std::lexicographical\_compare(set1.begin(), set1.end(), set2.begin(), set2.end())) {

std::cout << "set1 is lexicographically less than set2" << std::endl;

}

else {

std::cout << "set1 is lexicographically greater or equal to set2" << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **merge**

Пример использования алгоритма merge:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

#include <vector>

int main() {

std::unordered\_set<int> set1 = { 1, 3, 5 };

std::unordered\_set<int> set2 = { 2, 4, 6 };

std::vector<int> result;

std::merge(set1.begin(), set1.end(), set2.begin(), set2.end(), std::back\_inserter(result));

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **partial\_sort\_copy**

Пример использования алгоритма partial\_sort\_copy:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

#include <vector>

int main() {

std::unordered\_set<int> mySet = { 5, 2, 1, 4, 3 };

std::vector<int> result(3);

std::partial\_sort\_copy(mySet.begin(), mySet.end(), result.begin(), result.end());

for (const auto& element : result) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **equal\_range**

Пример использования алгоритма equal\_range:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 2, 3, 4, 4, 5 };

auto range = std::equal\_range(mySet.begin(), mySet.end(), 2);

for (auto it = range.first; it != range.second; ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **lower\_bound**

Пример использования алгоритма lower\_bound:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 2, 3, 4, 4, 5 };

auto it = std::lower\_bound(mySet.begin(), mySet.end(), 3);

if (it != mySet.end()) {

std::cout << "Element found: " << \*it << std::endl;

}

else {

std::cout << "Element not found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **upper\_bound**

Пример использования алгоритма upper\_bound:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_set<int> mySet = { 1, 2, 2, 3, 4, 4, 5 };

auto it = std::upper\_bound(mySet.begin(), mySet.end(), 2);

if (it != mySet.end()) {

std::cout << "Element found: " << \*it << std::endl;

}

else {

std::cout << "Element not found" << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

#### **Unordered\_multiset**

##### **adjacent\_find**

Пример использования алгоритма adjacent\_find:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> mySet = { 1, 2, 2, 3, 4, 4, 5 };

auto it = std::adjacent\_find(mySet.begin(), mySet.end());

if (it != mySet.end()) {

std::cout << "First adjacent pair with equal values: " << \*it << std::endl;

}

else {

std::cout << "No adjacent elements with equal values found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **count**

Пример использования алгоритма count:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> numbers = { 1, 2, 2, 3, 3, 3 };

int count = std::count(numbers.begin(), numbers.end(), 3);

std::cout << "Count: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **count\_if**

Пример использования алгоритма count­\_if:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

bool isEven(int number) {

return number % 2 == 0;

}

int main() {

std::unordered\_multiset<int> numbers = { 1, 2, 3, 4, 5, 6 };

int count = std::count\_if(numbers.begin(), numbers.end(), isEven);

std::cout << "Count of even numbers: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **equal**

Пример использования алгоритма equal:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> numbers1 = { 1, 2, 3, 4, 5 };

std::unordered\_multiset<int> numbers2 = { 1, 2, 3, 4, 5 };

bool areEqual = std::equal(numbers1.begin(), numbers1.end(), numbers2.begin(), numbers2.end());

if (areEqual) {

std::cout << "The sets are equal." << std::endl;

}

else {

std::cout << "The sets are not equal." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find**

Пример использования алгоритма find:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> mySet = { 1, 2, 2, 3, 4, 4, 5 };

int valueToFind = 3;

auto it = mySet.find(valueToFind);

if (it != mySet.end()) {

std::cout << "Element found: " << \*it << std::endl;

}

else {

std::cout << "Element not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find\_if**

Пример использования алгоритма find\_if:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

bool isEven(int number) {

return number % 2 == 0;

}

int main() {

std::unordered\_multiset<int> mySet = { 1, 2, 2, 3, 4, 4, 5 };

auto it = std::find\_if(mySet.begin(), mySet.end(), isEven);

if (it != mySet.end()) {

std::cout << "First even element found: " << \*it << std::endl;

}

else {

std::cout << "No even elements found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find\_first\_of**

Пример использования алгоритма find\_first\_of:

#include <iostream>

#include <algorithm>

#include <unordered\_set>

int main() {

std::unordered\_multiset<int> mySet = { 1, 2, 3, 4, 5, 6 };

std::unordered\_multiset<int> searchSet = { 5, 6, 7 };

auto result = std::find\_first\_of(mySet.begin(), mySet.end(), searchSet.begin(), searchSet.end());

if (result != mySet.end()) {

std::cout << "First element found at index: " << std::distance(mySet.begin(), result) << std::endl;

}

else {

std::cout << "No elements found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **for\_each**

Пример использования алгоритма for\_each:

#include <iostream>

#include <algorithm>

#include <unordered\_set>

void PrintElement(int value) {

std::cout << value << " ";

}

int main() {

std::unordered\_multiset<int> mySet = { 1, 2, 3, 4, 5 };

std::for\_each(mySet.begin(), mySet.end(), PrintElement);

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **mismatch**

Пример использования алгоритма mismatch:

#include <iostream>

#include <algorithm>

#include <unordered\_set>

int main() {

std::unordered\_multiset<int> set1 = { 1, 2, 3, 4, 5 };

std::unordered\_multiset<int> set2 = { 1, 2, 4, 4, 5 };

auto mismatchPair = std::mismatch(set1.begin(), set1.end(), set2.begin());

if (mismatchPair.first == set1.end() && mismatchPair.second == set2.end()) {

std::cout << "Sequences are equal." << std::endl;

}

else {

std::cout << "Mismatch found at index: " << std::distance(set1.begin(), mismatchPair.first) << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **seacrh**

Пример использования алгоритма search:

#include <iostream>

#include <algorithm>

#include <unordered\_set>

int main() {

std::unordered\_multiset<int> mySet = { 1, 2, 3, 4, 5, 6 };

std::unordered\_multiset<int> subSet = { 4, 5 };

auto result = std::search(mySet.begin(), mySet.end(), subSet.begin(), subSet.end());

if (result != mySet.end()) {

std::cout << "Subsequence found at index: " << std::distance(mySet.begin(), result) << std::endl;

}

else {

std::cout << "Subsequence not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **search\_n**

Пример использования алгоритма search\_n:

#include <iostream>

#include <algorithm>

#include <unordered\_set>

int main() {

std::unordered\_multiset<int> mySet = { 1, 2, 2, 2, 3, 4, 5 };

int value = 2;

int count = 3;

auto result = std::search\_n(mySet.begin(), mySet.end(), count, value);

if (result != mySet.end()) {

std::cout << "Sequence of " << count << " elements with value " << value << " found at index: " << std::distance(mySet.begin(), result) << std::endl;

}

else {

std::cout << "Sequence not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **max\_element**

Пример использования алгоритма max\_element:

#include <iostream>

#include <algorithm>

#include <unordered\_set>

int main() {

std::unordered\_multiset<int> mySet = { 5, 2, 8, 3, 1 };

auto maxValue = std::max\_element(mySet.begin(), mySet.end());

std::cout << "Max value: " << \*maxValue << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **min\_element**

Пример использования алгоритма min\_element:

#include <iostream>

#include <algorithm>

#include <unordered\_set>

int main() {

std::unordered\_multiset<int> mySet = { 5, 2, 8, 3, 1 };

auto minValue = std::min\_element(mySet.begin(), mySet.end());

std::cout << "Min value: " << \*minValue << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **accumulate**

Пример использования алгоритма accumulate:

#include <iostream>

#include <algorithm>

#include <numeric>

#include <unordered\_set>

int main() {

std::unordered\_multiset<int> mySet = { 1, 2, 3, 4, 5 };

int sum = std::accumulate(mySet.begin(), mySet.end(), 0);

std::cout << "Sum: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **adjacent\_difference**

Пример использования алгоритма adjacent\_difference:

#include <iostream>

#include <algorithm>

#include <numeric>

#include <unordered\_set>

#include <vector>

int main() {

std::unordered\_multiset<int> mySet = { 1, 2, 4, 7, 11 };

std::vector<int> differences(mySet.size());

std::adjacent\_difference(mySet.begin(), mySet.end(), differences.begin());

std::cout << "Adjacent differences: ";

for (const auto& diff : differences) {

std::cout << diff << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **inner\_product**

Пример использования алгоритма inner\_product:

#include <iostream>

#include <algorithm>

#include <numeric>

#include <unordered\_set>

int main() {

std::unordered\_multiset<int> set1 = { 1, 2, 3, 4, 5 };

std::unordered\_multiset<int> set2 = { 6, 7, 8, 9, 10 };

int result = std::inner\_product(set1.begin(), set1.end(), set2.begin(), 0);

std::cout << "Inner product: " << result << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **partial\_sum**

Пример использования алгоритма partial\_sum:

#include <iostream>

#include <unordered\_set>

#include <numeric>

#include <iterator>

int main() {

std::unordered\_multiset<int> mySet = { 1, 2, 3, 4, 5 };

std::unordered\_multiset<int> result;

std::partial\_sum(mySet.begin(), mySet.end(), std::inserter(result, result.end()));

std::cout << "Partial sums: ";

for (const auto& num : result) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **swap**

Пример использования алгоритма swap:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> set1 = { 1, 2, 3 };

std::unordered\_multiset<int> set2 = { 4, 5, 6 };

std::swap(set1, set2);

std::cout << "Set 1: ";

for (const auto& num : set1) {

std::cout << num << " ";

}

std::cout << std::endl;

std::cout << "Set 2: ";

for (const auto& num : set2) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **replacy\_copy**

Пример использования алгоритма replace\_copy:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> source = { 1, 2, 2, 3, 4, 5 };

std::unordered\_multiset<int> destination;

int old\_value = 2;

int new\_value = 10;

std::replace\_copy(source.begin(), source.end(), std::inserter(destination, destination.end()), old\_value, new\_value);

std::cout << "Destination: ";

for (const auto& element : destination) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **replace\_copy\_if**

Пример использования алгоритма replace\_copy\_if:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> source = { 1, 2, 2, 3, 4, 5 };

std::unordered\_multiset<int> destination;

int threshold = 3;

int new\_value = 10;

std::replace\_copy\_if(source.begin(), source.end(), std::inserter(destination, destination.end()),

[threshold](int element) { return element > threshold; }, new\_value);

std::cout << "Destination: ";

for (const auto& element : destination) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **remove**

Пример использования алгоритма remove:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> source = { 1, 2, 2, 3, 4, 5 };

std::unordered\_multiset<int> destination;

int threshold = 3;

int new\_value = 10;

std::replace\_copy\_if(source.begin(), source.end(), std::inserter(destination, destination.end()),

[threshold](int element) { return element > threshold; }, new\_value);

std::cout << "Destination: ";

for (const auto& element : destination) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **remove\_copy**

Пример использования алгоритма remove\_copy:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

int main() {

std::unordered\_multiset<int> sourceSet = { 1, 2, 3, 4, 5, 6 };

std::unordered\_multiset<int> destinationSet;

std::remove\_copy(sourceSet.begin(), sourceSet.end(), std::inserter(destinationSet, destinationSet.end()), 3);

std::cout << "Destination set:";

for (const auto& element : destinationSet) {

std::cout << " " << element;

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **remove\_copy\_if**

Пример использования алгоритма remove\_copy\_if:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

bool isEven(int num) {

return num % 2 == 0;

}

int main() {

std::unordered\_multiset<int> sourceSet = { 1, 2, 3, 4, 5, 6 };

std::unordered\_multiset<int> destinationSet;

std::remove\_copy\_if(sourceSet.begin(), sourceSet.end(), std::inserter(destinationSet, destinationSet.end()), isEven);

std::cout << "Destination set:";

for (const auto& element : destinationSet) {

std::cout << " " << element;

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **copy**

Пример использования алгоритма copy:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> sourceSet = { 1, 2, 2, 3, 4 };

std::unordered\_multiset<int> destinationSet;

std::copy(sourceSet.begin(), sourceSet.end(), std::inserter(destinationSet, destinationSet.end()));

std::cout << "Destination Set: ";

for (const auto& element : destinationSet) {

std::cout << element << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **reverse\_copy**

Пример использования алгоритма reverse\_copy:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <iterator>

int main() {

std::unordered\_multiset<int> numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

std::unordered\_multiset<int> reversedNumbers;

std::reverse\_copy(numbers.begin(), numbers.end(), std::inserter(reversedNumbers, reversedNumbers.begin()));

std::cout << "Reversed copy of unordered\_multiset: ";

for (const auto& number : reversedNumbers) {

std::cout << number << " ";

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

#include <vector>

int main() {

std::unordered\_multiset<int> numbers{ 1, 2, 3, 4, 5 };

std::vector<int> transformed\_numbers(numbers.size());

std::transform(numbers.begin(), numbers.end(), transformed\_numbers.begin(),

[](int num) { return num \* 2; });

for (const auto& num : transformed\_numbers) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **is\_sorted**

Пример использования алгоритма is\_sorted:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> numbers{ 1, 2, 3, 4, 5 };

bool is\_sorted = std::is\_sorted(numbers.begin(), numbers.end());

std::cout << "Is sorted: " << (is\_sorted ? "true" : "false") << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **lexicographical\_compare**

Пример использования алгоритма lexicographical\_compare:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> set1{ 1, 2, 3 };

std::unordered\_multiset<int> set2{ 1, 2, 4 };

bool is\_set1\_less = std::lexicographical\_compare(set1.begin(), set1.end(), set2.begin(), set2.end());

std::cout << "Set1 is less: " << (is\_set1\_less ? "true" : "false") << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **binary\_search**

Пример использования алгоритма binary\_search:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> numbers{ 1, 2, 3, 4, 5 };

bool found = std::binary\_search(numbers.begin(), numbers.end(), 3);

std::cout << "Found: " << (found ? "true" : "false") << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **equal\_range**

Пример использования алгоритма equal\_range:

#include <iostream>

#include <unordered\_set>

#include <algorithm>

int main() {

std::unordered\_multiset<int> mySet = { 4, 2, 1, 5, 3, 2 };

std::cout << "Elements in the set: ";

for (const auto& element : mySet) {

std::cout << element << " ";

}

std::cout << std::endl;

int target = 2;

auto range = mySet.equal\_range(target);

std::cout << "Range of elements with value " << target << ": ";

for (auto it = range.first; it != range.second; ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

## **Unordered\_map**

#### **Unordered\_map**

##### **adjacent\_find**

Пример использования алгоритма adjacent\_find:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "cherry"},

{4, "banana"},

{5, "apple"}

};

auto it = std::adjacent\_find(myMap.begin(), myMap.end());

if (it != myMap.end()) {

std::cout << "Adjacent elements with the same value found: " << it->first << " and " << std::next(it)->first << std::endl;

}

else {

std::cout << "No adjacent elements with the same value found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **count**

Пример использования алгоритма count:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<char, int> myMap = {

{'a', 1},

{'b', 2},

{'c', 3},

{'d', 2},

{'e', 1}

};

int count = std::count(myMap.begin(), myMap.end(), std::pair<const char, int>('a', 1));

std::cout << "Number of occurrences: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **count\_if**

Пример использования алгоритма count­\_if:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "cherry"},

{4, "durian"},

{5, "apple"}

};

int count = std::count\_if(myMap.begin(), myMap.end(), [](const auto& pair) {

return pair.second == "apple";

});

std::cout << "Number of occurrences of 'apple': " << count << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **equal**

Пример использования алгоритма equal:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, std::string> map1 = {

{1, "apple"},

{2, "banana"},

{3, "cherry"}

};

std::unordered\_map<int, std::string> map2 = {

{1, "apple"},

{2, "banana"},

{3, "cherry"}

};

bool isEqual = std::equal(map1.begin(), map1.end(), map2.begin());

if (isEqual) {

std::cout << "The maps are equal." << std::endl;

}

else {

std::cout << "The maps are not equal." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find**

Пример использования алгоритма find:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "cherry"}

};

auto it = std::find(myMap.begin(), myMap.end(), std::pair<const int, std::string>(2, "banana"));

if (it != myMap.end()) {

std::cout << "Element found: " << it->first << " - " << it->second << std::endl;

}

else {

std::cout << "Element not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find\_if**

Пример использования алгоритма find\_if:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "cherry"}

};

auto it = std::find\_if(myMap.begin(), myMap.end(), [](const auto& pair) {

return pair.second == "banana";

});

if (it != myMap.end()) {

std::cout << "Element found: " << it->first << " - " << it->second << std::endl;

}

else {

std::cout << "Element not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find\_end**

Пример использования алгоритма find\_end:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

#include <vector>

int main() {

std::unordered\_map<int, std::string> map1 = {

{1, "apple"},

{2, "banana"},

{3, "cherry"},

{4, "apple"},

{5, "banana"},

{6, "cherry"}

};

std::unordered\_map<int, std::string> map2 = {

{4, "apple"},

{5, "banana"},

{6, "cherry"}

};

auto it = std::find\_end(map1.begin(), map1.end(), map2.begin(), map2.end());

if (it != map1.end()) {

std::cout << "Last occurrence of map2 found starting at: " << it->first << " - " << it->second << std::endl;

}

else {

std::cout << "map2 not found in map1." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **for\_each**

Пример использования алгоритма for\_each:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

void printPair(const std::pair<const int, std::string>& pair) {

std::cout << pair.first << " - " << pair.second << std::endl;

}

int main() {

std::unordered\_map<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "cherry"}

};

std::for\_each(myMap.begin(), myMap.end(), printPair);

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **mismatch**

Пример использования алгоритма mismatch:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, std::string> map1 = {

{1, "apple"},

{2, "banana"},

{3, "cherry"}

};

std::unordered\_map<int, std::string> map2 = {

{1, "apple"},

{2, "orange"},

{3, "cherry"}

};

auto mismatchPair = std::mismatch(map1.begin(), map1.end(), map2.begin());

if (mismatchPair.first != map1.end() && mismatchPair.second != map2.end()) {

std::cout << "First mismatch found at: " << mismatchPair.first->first << " - " << mismatchPair.first->second << std::endl;

std::cout << "Corresponding element in map2: " << mismatchPair.second->first << " - " << mismatchPair.second->second << std::endl;

}

else {

std::cout << "Maps are identical." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **search**

Пример использования алгоритма search:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

#include <vector>

int main() {

std::unordered\_map<int, std::string> map1 = {

{1, "apple"},

{2, "banana"},

{3, "cherry"},

{4, "apple"},

{5, "banana"},

{6, "cherry"}

};

std::unordered\_map<int, std::string> map2 = {

{2, "banana"},

{3, "cherry"}

};

auto it = std::search(map1.begin(), map1.end(), map2.begin(), map2.end());

if (it != map1.end()) {

std::cout << "First occurrence of map2 found starting at: " << it->first << " - " << it->second << std::endl;

}

else {

std::cout << "map2 not found in map1." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **search\_n**

Пример использования алгоритма search­\_n:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "apple"},

{4, "banana"},

{5, "apple"}

};

int count = 2;

auto it = std::search\_n(myMap.begin(), myMap.end(), count, std::pair<const int, std::string>(1, "apple"));

if (it != myMap.end()) {

std::cout << "First occurrence of " << count << " consecutive 'apple' elements found starting at: " << it->first << " - " << it->second << std::endl;

}

else {

std::cout << "No occurrence of " << count << " consecutive 'apple' elements found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **max\_element**

Пример использования алгоритма max\_element:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, int> myMap = {

{1, 5},

{2, 10},

{3, 3},

{4, 8},

{5, 2}

};

auto maxElement = std::max\_element(myMap.begin(), myMap.end(), [](const auto& pair1, const auto& pair2) {

return pair1.second < pair2.second;

});

if (maxElement != myMap.end()) {

std::cout << "Maximum value: " << maxElement->second << std::endl;

}

else {

std::cout << "The map is empty." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **min\_element**

Пример использования алгоритма min\_element:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, int> myMap = {

{1, 5},

{2, 10},

{3, 3},

{4, 8},

{5, 2}

};

auto minElement = std::min\_element(myMap.begin(), myMap.end(), [](const auto& pair1, const auto& pair2) {

return pair1.second < pair2.second;

});

if (minElement != myMap.end()) {

std::cout << "Minimum value: " << minElement->second << std::endl;

}

else {

std::cout << "The map is empty." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **accumulate**

Пример использования алгоритма accumulate:

#include <iostream>

#include <unordered\_map>

#include <numeric>

int main() {

std::unordered\_map<int, int> myMap = {

{1, 5},

{2, 10},

{3, 3},

{4, 8},

{5, 2}

};

int sum = std::accumulate(myMap.begin(), myMap.end(), 0, [](int currentSum, const auto& pair) {

return currentSum + pair.second;

});

std::cout << "Sum of values: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **inner\_product**

Пример использования алгоритма inner\_product:

#include <iostream>

#include <unordered\_map>

#include <numeric>

int main() {

std::unordered\_map<int, int> myMap1 = {

{1, 5},

{2, 10},

{3, 3},

{4, 8},

{5, 2}

};

std::unordered\_map<int, int> myMap2 = {

{1, 2},

{2, 4},

{3, 6},

{4, 8},

{5, 10}

};

int result = std::inner\_product(myMap1.begin(), myMap1.end(), myMap2.begin(), 0, std::plus<>(), [](const auto& pair1, const auto& pair2) {

return pair1.second \* pair2.second;

});

std::cout << "Inner product: " << result << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **swap**

Пример использования алгоритма swap:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, int> myMap = {

{1, 10},

{2, 20},

{3, 30}

};

int key1 = 1;

int key2 = 2;

auto it1 = myMap.find(key1);

auto it2 = myMap.find(key2);

if (it1 != myMap.end() && it2 != myMap.end()) {

std::swap(it1->second, it2->second);

std::cout << "Values swapped successfully." << std::endl;

}

else {

std::cout << "One or both keys not found in the map." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **remove\_copy\_if**

Пример использования алгоритма remove\_copy\_if:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

#include <iterator>

int main() {

std::unordered\_map<int, int> myMap = {

{1, 10},

{2, 20},

{3, 30},

{4, 10},

{5, 40}

};

int threshold = 25;

std::unordered\_map<int, int> newMap;

std::remove\_copy\_if(myMap.begin(), myMap.end(), std::inserter(newMap, newMap.end()), [&threshold](const auto& pair) {

return pair.second > threshold;

});

for (const auto& pair : newMap) {

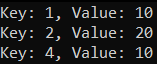
std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **copy**

Пример использования алгоритма copy:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

#include <iterator>

int main() {

std::unordered\_map<int, int> sourceMap = {

{1, 10},

{2, 20},

{3, 30},

{4, 40},

{5, 50}

};

std::unordered\_map<int, int> destinationMap;

std::copy(sourceMap.begin(), sourceMap.end(), std::inserter(destinationMap, destinationMap.end()));

for (const auto& pair : destinationMap) {

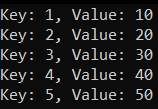
std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **fill\_n**

Пример использования алгоритма fill\_n:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, int> myMap;

std::fill\_n(std::inserter(myMap, myMap.end()), 5, std::pair<int, int>(10, 20));

for (const auto& pair : myMap) {

std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **genetate\_n**

Пример использования алгоритма generate\_n:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

#include <random>

int main() {

std::unordered\_map<int, int> myMap;

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(1, 100);

std::generate\_n(std::inserter(myMap, myMap.end()), 5, [&dis, &gen]() { return std::make\_pair(dis(gen), dis(gen)); });

for (const auto& pair : myMap) {

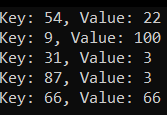
std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **reverse\_copy**

Пример использования алгоритма reverse\_copy:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

#include <iterator>

int main() {

std::unordered\_map<int, int> myMap = {

{1, 10},

{2, 20},

{3, 30},

{4, 40},

{5, 50}

};

std::unordered\_map<int, int> reversedMap;

std::reverse\_copy(myMap.begin(), myMap.end(), std::inserter(reversedMap, reversedMap.end()));

for (const auto& pair : reversedMap) {

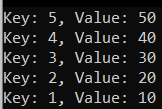
std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **is\_sorted**

Пример использования алгоритма is\_sorted:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, int> myMap = {

{1, 10},

{2, 20},

{3, 30},

{4, 40},

{5, 50}

};

bool isSorted = std::is\_sorted(myMap.begin(), myMap.end());

if (isSorted) {

std::cout << "The map is sorted." << std::endl;

}

else {

std::cout << "The map is not sorted." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **lexicographical\_compare**

Пример использования алгоритма lexicographical\_compare:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, int> map1 = {

{1, 10},

{2, 20},

{3, 30}

};

std::unordered\_map<int, int> map2 = {

{1, 10},

{2, 20},

{4, 40}

};

bool isLess = std::lexicographical\_compare(map1.begin(), map1.end(), map2.begin(), map2.end());

if (isLess) {

std::cout << "map1 is lexicographically less than map2." << std::endl;

}

else {

std::cout << "map1 is not lexicographically less than map2." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **equal\_range**

Пример использования алгоритма equal\_range:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, int> myMap = {

{1, 10},

{2, 20},

{3, 30},

{3, 40},

{5, 50}

};

int key = 3;

auto range = myMap.equal\_range(key);

std::cout << "Elements with key " << key << ": " << std::endl;

for (auto it = range.first; it != range.second; ++it) {

std::cout << "Key: " << it->first << ", Value: " << it->second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **lower\_bound**

Пример использования алгоритма lower\_bound:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_map<int, int> myMap = {

{1, 10},

{2, 20},

{3, 30},

{4, 40},

{5, 50}

};

int key = 3;

auto it = std::lower\_bound(myMap.begin(), myMap.end(), key,

[](const auto& pair1, const auto& pair2) {

return pair1.first < pair2;

});

if (it != myMap.end() && it->first == key) {

std::cout << "Key " << key << " found in the map." << std::endl;

}

else {

std::cout << "Key " << key << " not found in the map." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

#### **Unordered\_multimap**

##### **count\_if**

Пример использования алгоритма count­\_if:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "apple"},

{4, "banana"},

{5, "grape"}

};

int count = std::count\_if(myMap.begin(), myMap.end(),

[](const auto& pair) {

return pair.second == "apple";

}

);

std::cout << "Count of 'apple' elements: " << count << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **equal**

Пример использования алгоритма equal:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, std::string> myMap1 = {

{1, "apple"},

{2, "banana"},

{3, "apple"}

};

std::unordered\_multimap<int, std::string> myMap2 = {

{1, "apple"},

{2, "banana"},

{3, "apple"}

};

bool isEqual = std::equal(myMap1.begin(), myMap1.end(), myMap2.begin());

if (isEqual) {

std::cout << "The two maps are equal." << std::endl;

}

else {

std::cout << "The two maps are not equal." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find\_if**

Пример использования алгоритма find\_if:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "apple"},

{4, "banana"},

{5, "grape"}

};

auto it = std::find\_if(myMap.begin(), myMap.end(),

[](const auto& pair) {

return pair.second == "banana";

}

);

if (it != myMap.end()) {

std::cout << "Element found: " << it->second << std::endl;

}

else {

std::cout << "Element not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find\_end**

Пример использования алгоритма find\_end:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "apple"},

{4, "banana"},

{5, "grape"}

};

std::unordered\_multimap<int, std::string> subMap = {

{3, "apple"},

{4, "banana"}

};

auto it = std::find\_end(myMap.begin(), myMap.end(), subMap.begin(), subMap.end());

if (it != myMap.end()) {

std::cout << "Submap found at position: " << std::distance(myMap.begin(), it) << std::endl;

}

else {

std::cout << "Submap not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **find\_first\_of**

Пример использования алгоритма find\_first\_of:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "apple"},

{4, "banana"},

{5, "grape"}

};

std::unordered\_multimap<int, std::string> searchMap = {

{6, "banana"},

{7, "grape"}

};

auto it = std::find\_first\_of(myMap.begin(), myMap.end(), searchMap.begin(), searchMap.end(),

[](const auto& pair1, const auto& pair2) {

return pair1.second == pair2.second;

}

);

if (it != myMap.end()) {

std::cout << "First element found: " << it->second << std::endl;

}

else {

std::cout << "No elements found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **for\_each**

Пример использования алгоритма for\_each:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

void printPair(const std::pair<const int, std::string>& pair) {

std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;

}

int main() {

std::unordered\_multimap<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "apple"},

{4, "banana"},

{5, "grape"}

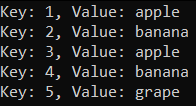
};

std::for\_each(myMap.begin(), myMap.end(), printPair);

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **mismatch**

Пример использования алгоритма mismatch:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, std::string> myMap1 = {

{1, "apple"},

{2, "banana"},

{3, "apple"}

};

std::unordered\_multimap<int, std::string> myMap2 = {

{1, "apple"},

{2, "banana"},

{3, "grape"}

};

auto mismatchPair = std::mismatch(myMap1.begin(), myMap1.end(), myMap2.begin());

if (mismatchPair.first != myMap1.end() || mismatchPair.second != myMap2.end()) {

std::cout << "Mismatch found: "

<< mismatchPair.first->second << " != " << mismatchPair.second->second << std::endl;

}

else {

std::cout << "No mismatches found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **search**

Пример использования алгоритма search:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "apple"},

{4, "banana"},

{5, "grape"}

};

std::unordered\_multimap<int, std::string> searchMap = {

{3, "apple"},

{4, "banana"}

};

auto it = std::search(myMap.begin(), myMap.end(), searchMap.begin(), searchMap.end());

if (it != myMap.end()) {

std::cout << "Submap found at position: " << std::distance(myMap.begin(), it) << std::endl;

}

else {

std::cout << "Submap not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **search\_n**

Пример использования алгоритма search­\_n:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, std::string> myMap = {

{1, "apple"},

{2, "banana"},

{3, "apple"},

{4, "banana"},

{5, "grape"},

{6, "apple"}

};

int count = 2;

std::string value = "apple";

auto it = std::search\_n(myMap.begin(), myMap.end(), count, std::make\_pair(0, value),

[](const auto& pair1, const auto& pair2) {

return pair1.second == pair2.second;

}

);

if (it != myMap.end()) {

std::cout << "Sequence of " << count << " '" << value << "' elements found at position: "

<< std::distance(myMap.begin(), it) << std::endl;

}

else {

std::cout << "Sequence not found." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **max\_element**

Пример использования алгоритма max\_element:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, int> myMap = {

{1, 10},

{2, 20},

{3, 30},

{4, 40},

{5, 50}

};

auto maxValue = std::max\_element(myMap.begin(), myMap.end(),

[](const auto& pair1, const auto& pair2) {

return pair1.second < pair2.second;

}

);

if (maxValue != myMap.end()) {

std::cout << "Maximum value: " << maxValue->second << std::endl;

}

else {

std::cout << "Map is empty." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **min\_element**

Пример использования алгоритма min\_element:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, int> myMap = {

{1, 10},

{2, 20},

{3, 30},

{4, 40},

{5, 50}

};

auto minValue = std::min\_element(myMap.begin(), myMap.end(),

[](const auto& pair1, const auto& pair2) {

return pair1.second < pair2.second;

}

);

if (minValue != myMap.end()) {

std::cout << "Minimum value: " << minValue->second << std::endl;

}

else {

std::cout << "Map is empty." << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **accumulate**

Пример использования алгоритма accumulate:

#include <iostream>

#include <unordered\_map>

#include <numeric>

int main() {

std::unordered\_multimap<int, int> myMap = {

{1, 10},

{2, 20},

{3, 30},

{4, 40},

{5, 50}

};

int sum = std::accumulate(myMap.begin(), myMap.end(), 0,

[](int currentSum, const auto& pair) {

return currentSum + pair.second;

}

);

std::cout << "Sum: " << sum << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **swap**

Пример использования алгоритма swap:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, int> map1 = {

{1, 10},

{2, 20},

{3, 30}

};

std::unordered\_multimap<int, int> map2 = {

{4, 40},

{5, 50}

};

std::swap(map1, map2);

std::cout << "map1 after swap:" << std::endl;

for (const auto& pair : map1) {

std::cout << pair.first << ": " << pair.second << std::endl;

}

std::cout << "map2 after swap:" << std::endl;

for (const auto& pair : map2) {

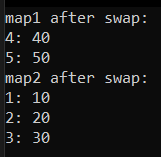
std::cout << pair.first << ": " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **copy**

Пример использования алгоритма copy:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

#include <vector>

int main() {

std::unordered\_multimap<int, int> myMap = {

{1, 10},

{2, 20},

{3, 30},

{4, 40},

{5, 50}

};

std::vector<std::pair<int, int>> target(myMap.size());

std::copy(myMap.begin(), myMap.end(), target.begin());

std::cout << "Copied elements:" << std::endl;

for (const auto& pair : target) {

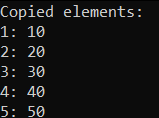
std::cout << pair.first << ": " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **transform**

Пример использования алгоритма transform:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

#include <functional>

int main() {

std::unordered\_multimap<int, int> myMap = {

{1, 10},

{2, 20},

{3, 30},

{4, 40},

{5, 50}

};

std::unordered\_multimap<int, int> transformedMap;

std::transform(myMap.begin(), myMap.end(), std::inserter(transformedMap, transformedMap.end()),

[](const auto& pair) {

return std::make\_pair(pair.first + 10, pair.second \* 2); // Transform keys and values

});

std::cout << "Transformed elements:" << std::endl;

for (const auto& pair : transformedMap) {

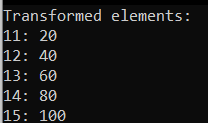
std::cout << pair.first << ": " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **lexicographical\_compare**

Пример использования алгоритма lexicographical\_compare:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, int> myMap1 = {

{1, 10},

{2, 20},

{3, 30}

};

std::unordered\_multimap<int, int> myMap2 = {

{1, 10},

{2, 20},

{4, 40}

};

bool isLess = std::lexicographical\_compare(myMap1.begin(), myMap1.end(), myMap2.begin(), myMap2.end());

std::cout << "Is myMap1 lexicographically less than myMap2? " << std::boolalpha << isLess << std::endl;

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **merge**

Пример использования алгоритма merge:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

#include <iterator>

int main() {

std::unordered\_multimap<int, int> myMap1 = {

{1, 10},

{2, 20},

{3, 30}

};

std::unordered\_multimap<int, int> myMap2 = {

{4, 40},

{5, 50},

{6, 60}

};

std::unordered\_multimap<int, int> mergedMap;

std::merge(myMap1.begin(), myMap1.end(), myMap2.begin(), myMap2.end(), std::inserter(mergedMap, mergedMap.end()));

std::cout << "Merged elements:" << std::endl;

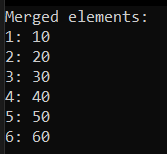
for (const auto& pair : mergedMap) {

std::cout << pair.first << ": " << pair.second << std::endl;

}

return 0;

}

Результат работы программы:

[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)

##### **equal\_range**

Пример использования алгоритма equal\_range:

#include <iostream>

#include <unordered\_map>

#include <algorithm>

int main() {

std::unordered\_multimap<int, int> myMap = {

{5, 50},

{3, 30},

{1, 10},

{4, 40},

{2, 20}

};

int key = 3;

auto range = myMap.equal\_range(key);

std::cout << "Elements with key " << key << ":" << std::endl;

for (auto it = range.first; it != range.second; ++it) {

std::cout << it->first << ": " << it->second << std::endl;

}

return 0;

}

Результат работы программы:



[<Неупорядоченные ассоциативные контейнеры>](#_Неупорядоченные_ассоциативные_конте)