Общие методы для всех контейнеров	
`container<type> a(b)`	Создаёт новый экземпляр контейнера типа container на основе уже существующего контейнера `b`.
`a.begin()`	Возвращает итератор, указывающий на первый элемент контейнера `a`.
`a.end()`	Возвращает итератор, указывающий на элемент, следующий за последним в контейнере `a`.
`a.size()`	Возвращает размер контейнера `a`, то есть, количество элементов находящихся в нём.
`a.max_size()`	Возвращает максимальный размер, который может иметь контейнер `a`.
`a.empty()`	Определяет, пуст ли контейнер `a`. Эквивалентно `a.size() == 0`.
`a.swap(b)`	Обменивает поэлементно содержимое контейнеров `a` и `b`. Эквивалентно `swap(a,b)`.
vector	
`vector<type> a(n, t)`	Создаёт контейнер типа vector, содержащий `n` элементов со значением `t`.
`vector<type> a(n)`	Создает контейнер типа vector, содержащий `n` элементов инициализированных значением по умолчанию.
`vector<type> a()`	Создаёт пустой контейнер типа vector.
`vector<type> a(i, j)`	Создает последовательность, которая является копией диапазона `[i,j)`, где `i` – итератор на первый элемент диапазона, а `j` – итератор на элемент следующий за последним элементом диапазона
`a.push_back(t)`	Вставляет копию значения `t` в конец контейнера.
`a.pop_back()`	Возвращает итератор на копию значения последнего элемента, который из контейнера удаляется.
`a.back()`	Возвращает значение последнего элемента контейнера `a`.
`a.insert(p, t)`	Вставляет копию значения `t` перед элементом, на который указывает итератор `p`.
`a.insert(p, n, t)`	Вставляет `n` копий значения `t` перед элементом, на который указывает итератор `p`.
`a.insert(p, i, j)`	Вставляет копию диапазона `[i,j)` перед элементом, на который указывает итератор `p`.
`a.erase(p)`	Разрушает элемент, на который указывает итератор `p` и удаляет его из контейнера `a`.
`a.erase(p,q)`	Разрушает элементы контейнера `a` из диапазона `[p,q)` и удаляет их из контейнера `a`.
`a.clear()`	Разрушает все элементы и удаляет их из контейнера `a`. Эквивалентно `a.erase(a.begin(), a.end())`.
`a.resize(n, t)`	Изменяет контейнер так, что он содержит ровно `n` элементов. Если необходимо дополнить контейнер `a`, то вставляются копии значения `t`.
`a.resize(n)`	Изменяет контейнер так, что он содержит ровно `n` элементов. Если необходимо дополнить контейнер `a`, то вставляются значения по умолчанию.
`a.rbegin()`	Возвращает итератор, указывающий на последний элемент контейнера `a`.
`a.rend()`	Возвращает итератор, указывающий на элемент, расположенный перед первым элементом контейнера `a`.
`a[n]`	Возвращает n-ый элемент от начала контейнера a.
`a.capacity()`	Возвращает количество элементов, для которых зарезервирована.память.
`a.reserve(n)`	Выделяет дополнительную память для последующего размещения новых элементов.
deque	
`deque<type> a(n, t)`	Создаёт контейнер типа deque, содержащий `n` элементов со значением `t`.
`deque <type> a(n)`	Создает контейнер типа deque, содержащий `n` элементов инициализированных значением по умолчанию.
`deque <type> a()`	Создаёт пустой контейнер типа deque.
`deque <type> a(i, j)`	Создает последовательность, которая является копией диапазона `[i,j)`, где `i` – итератор на первый элемент диапазона, а `j` – итератор на элемент следующий за последним элементом диапазона
`a.front()`	Возвращает значение первого элемента контейнера `a`.
`a.push_front(t)`	Вставляет копию значения `t` в начало контейнера.
`a.pop_front()`	Возвращает итератор на копию значения первого элемента, который из контейнера удаляется.
`a.back()`	Возвращает значение последнего элемента контейнера `a`.
`a.push_back(t)`	Вставляет копию значения `t` в конец контейнера.
`a.pop_back()`	Возвращает итератор на копию значения последнего элемента, который из контейнера удаляется.
`a.insert(p, t)`	Вставляет копию значения `t` перед элементом, на который указывает итератор `p`.
`a.insert(p, n, t)`	Вставляет `n` копий значения `t` перед элементом, на который указывает итератор `p`.
`a.insert(p, i, j)`	Вставляет копию диапазона `[i,j)` перед элементом, на который указывает итератор `p`.
`a.erase(p)`	Разрушает элемент, на который указывает итератор `p` и удаляет его из контейнера `a`.
`a.erase(p,q)`	Разрушает элементы контейнера `a` из диапазона `[p,q)` и удаляет их из контейнера `a`.
`a.clear()`	Разрушает все элементы и удаляет их из контейнера `a`. Эквивалентно `a.erase(a.begin(), a.end())`.
`a.resize(n, t)`	Изменяет контейнер так, что он содержит ровно `n` элементов. Если необходимо дополнить контейнер `a`, то вставляются копии значения `t`.
`a.resize(n)`	Изменяет контейнер так, что он содержит ровно `n` элементов. Если необходимо дополнить контейнер `a`, то вставляются значения по умолчанию.
`a.rbegin()`	Возвращает итератор, указывающий на последний элемент контейнера `a`.
`a.rend()`	Возвращает итератор, указывающий на элемент, расположенный перед первым элементом контейнера `a`.
`a[n]`	Возвращает n-ый элемент от начала контейнера a.
list	
`list<type> a(n, t)`	Создаёт контейнер типа list, содержащий `n` элементов со значением `t`.
`list <type> a(n)`	Создает контейнер типа list, содержащий `n` элементов инициализированных значением по умолчанию.
`list <type> a()`	Создаёт пустой контейнер типа list.
`list <type> a(i, j)`	Создает последовательность, которая является копией диапазона `[i,j)`, где `i` – итератор на первый элемент диапазона, а `j` – итератор на элемент следующий за последним элементом диапазона
`a.front()`	Возвращает значение первого элемента контейнера `a`.
`a.push_front(t)`	Вставляет копию значения `t` в начало контейнера.
`a.pop_front()`	Возвращает итератор на копию значения первого элемента, который из контейнера удаляется.
`a.back()`	Возвращает значение последнего элемента контейнера `a`.
`a.push_back(t)`	Вставляет копию значения `t` в конец контейнера.
`a.pop_back()`	Возвращает итератор на копию значения последнего элемента, который из контейнера удаляется.
`a.insert(p, t)`	Вставляет копию значения `t` перед элементом, на который указывает итератор `p`.
`a.insert(p, n, t)`	Вставляет `n` копий значения `t` перед элементом, на который указывает итератор `p`.
`a.insert(p, i, j)`	Вставляет копию диапазона `[i,j)` перед элементом, на который указывает итератор `p`.
`a.erase(p)`	Разрушает элемент, на который указывает итератор `p` и удаляет его из контейнера `a`.
`a.erase(p,q)`	Разрушает элементы контейнера `a` из диапазона `[p,q)` и удаляет их из контейнера `a`.

a.clear()	Разрушает все элементы и удаляет их из контейнера a. Эквивалентно a.erase(a.begin(), a.end()).
a.resize(n, t)	Изменяет контейнер так, что он содержит ровно n элементов. Если необходимо дополнить контейнер a, то вставляются копии значения t.
a.resize(n)	Изменяет контейнер так, что он содержит ровно n элементов. Если необходимо дополнить контейнер a, то вставляются значения по умолчанию.
a.rbegin()	Возвращает итератор, указывающий на последний элемент контейнера a.
a.rend()	Возвращает итератор, указывающий на элемент, расположенный перед первым элементом контейнера a.
a.splice(p, x)	Вставляет всех элементов списка x в список a перед элементом, на который указывает итератор p, а сами элементы из списка x удаляет.
a.splice(p, x, i)	Вставляет элемент, на который указывает итератор i, из списка x в список a перед элементом, на который указывает итератор p.
a.splice(p, x, f, l)	Вставляет элементы из диапазона [f, l) списка x в список a перед элементом, на который указывает итератор p.
a.remove(val)	Удаляет из списка a все элементы, значение которых равно val.
a.remove_if(p)	Удаляет из списка a все элементы, значение которых удовлетворяет предикату p.
a.unique()	Удаляет из списка a все элементы кроме первого в каждой последовательной группе элементов с одинаковыми значениями.
a.unique(p)	Удаляет из списка a все элементы кроме первого в каждой последовательной группе элементов, удовлетворяющих предикату p.
a.merge(x)	Сортирует два списка a и x, а затем объединяет их в один отсортированный список, который копирует в a. Элементы из списка x удаляет.
a.merge(x, c)	Сортирует два списка a и x с использованием функции-объекта c, а затем объединяет их в один отсортированный список, который копирует в a. Элементы из списка x удаляет.
a.reverse()	Меняет порядок следования элементов в списке a на обратный.
a.sort()	Сортирует список a, используя для сравнения элементов operator<.
a.sort(c)	Сортирует список a, используя для сравнения элементов функцию-объект c.
stack	
a.empty()	Возвращает true, если стек a не содержит элементов, а иначе false.
a.size()	Возвращает количество элементов, содержащихся в стеке a.
a.top()	Возвращает ссылку на элемент, находящийся на вершине стека a.
a.push(t)	Помещает копию значения t на вершину стека a.
a.pop()	Удаляет элемент, находящийся на вершине стека a.
queue	
a.empty()	Возвращает true, если очередь a не содержит элементов, а иначе false.
a.size()	Возвращает количество элементов, содержащихся в очереди a.
a.front()	Возвращает ссылку на элемент, который стоит в очереди a первым.
a.back()	Возвращает ссылку на элемент, который стоит в очереди a последним.
a.push(t)	Помещает копию значения t в конец очереди a.
a.pop()	Удаляет элемент, который стоит в очереди a первым.

Общие методы для всех ассоциативных массивов	
a.lower_bound(k)	Возвращает итератор, указывающий на первый элемент, чей ключ меньше чем k. Возвращает a.end(), если такого элемента не существует.
a.upper_bound(k)	Возвращает итератор, указывающий на первый элемент, чей ключ больше чем k. Возвращает a.end(), если такого элемента не существует.
a.erase(k)	Разрушает все элементы, у которых ключ имеет значение k, и удаляет их из контейнера a. Возвращает количество элементов, которые были удалены.
a.erase(p)	Разрушает элемент, на который указывает итератор p, и удаляет его из контейнера a.
a.erase(p, q)	Разрушает элементы контейнера из диапазона [p,q) и удаляет их из контейнера a.
a.clear()	Разрушает все элементы и удаляет их из контейнера a. Эквивалентно a.erase(a.begin(), a.end()).
a.find(k)	Возвращает итератор, указывающий на элемент, у которого ключ имеет значение k, или a.end(), если не существует такого элемента.
a.count(k)	Возвращает количество элементов в контейнере a, у которых ключи имеет значение k.
a.equal_range(k)	Возвращает пару P такую, что [P.first, P.second) – диапазон содержащий все элементы из контейнера a, у которых ключи имеют значение k. Если нет элементов, имеющих ключ со значением k, то возвращается пустой диапазон.
set	
set<key> a(i, j)	Создаёт контейнер, который содержит элементы из диапазона [i,j) с уникальными ключами типа key.
a.insert(t)	Вставляет значение t в контейнер a, если он не содержит элемента с таким же ключом как у t. Возвращает пару P, такую, что P.first – итератор указывающий на элемент, чей ключ имеет значение такое же как у t. P.second – логическое значение равное true если t был вставлен в контейнер a, и false если значение с таким же ключом как у t уже находилось в контейнере.
a.insert(p, t)	Вставляет копию значения t в контейнер a, если он не содержит элемента с таким же ключом как у t. Итератор p является рекомендацией, указывающей на элемент с которого следует начинать поиск. Возвращает разыменованный итератор указывающий на элемент с таким же ключом как у t.
a.insert(i, j)	Вставляет значения из диапазона [i, j) в контейнер a, если элементы с подобными ключами не содержатся в контейнере.
map	
map<key,value> a(i, j)	Создаёт контейнер, который содержит элементы из диапазона [i,j) с уникальными ключами типа key.
a.insert(t)	Вставляет значение t в контейнер a, если он не содержит элемента с таким же ключом как у t. Возвращает пару P, такую, что P.first – итератор указывающий на элемент, чей ключ имеет значение такое же как у t. P.second – логическое значение равное true если t был вставлен в контейнер a, и false если значение с таким же ключом как у t уже находилось в контейнере.
a.insert(i, j)	Вставляет значения из диапазона [i, j) в контейнер a, если элементы с подобными ключами не содержатся в

	контейнере.
`a.insert(p, t)`	Вставляет копию значения `t` в контейнер `a`, если он не содержит элемента с таким же ключом как у `t`. Итератор `p` является рекомендацией, указывающей на элемент с которого следует начинать поиск. Возвращает разыменованный итератор указывающий на элемент с таким же ключом как у `t`.
`a[k]`	Возвращает ссылку на объект, который ассоциирован с заданным ключом `k`. Если в карте `a` элемента с таким ключом не существует, то вставляет элемент с ключом `k` и значением по умолчанию.
multiset	
`multiset<key> a(i, j)`	Создаёт контейнер, который содержит все элементы из диапазона `[i,j)`.
`a.insert(t)`	Вставляет копию значения `t` в контейнер `a`.
`a.insert(i, j)`	Вставляет копии всех значений из диапазона `[i, j)` в контейнер `a`.
`a.insert(p, t)`	Вставляет копию значения `t` в контейнер `a`. Итератор `p` является рекомендацией, указывающей на элемент с которого следует начинать поиск. Возвращает разыменованный итератор указывающий на элемент с таким же ключом как у `t`.
multimap	
`multimap<key,value> a(i, j)`	Создаёт контейнер, который содержит все элементы из диапазона `[i,j)`
`a.insert(t)`	Вставляет копию значения `t` в контейнер `a`.
`a.insert(i, j)`	Вставляет копии всех значений из диапазона `[i, j)` в контейнер `a`.
`a.insert(p, t)`	Вставляет копию значения `t` в контейнер `a`. Итератор `p` является рекомендацией, указывающей на элемент с которого следует начинать поиск. Возвращает разыменованный итератор указывающий на элемент с таким же ключом как у `t`.

Алгоритмы, не изменяющие последовательность

```
ForwardIterator adjacent_find(ForwardIterator first, ForwardIterator last);
ForwardIterator adjacent_find(ForwardIterator first, ForwardIterator last, BinaryPredicate comp);
```

Осуществляет поиск и возвращает первый найденный итератор i, указывающий на элемент диапазона [first,last), значение которого равно следующему элементу или удовлетворяет предикату comp(*i,*(i+1)).

```
difference_type count(InputIterator first, InputIterator last, const EqualityComparable& value);
difference_type count_if(InputIterator first, InputIterator last, Predicate pred);
```

Определяет и возвращает количество элементов диапазона [first,last), значение которых равно value или удовлетворяет предикату pred.

```
bool equal(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2);
bool equal(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, BinaryPredicate comp);
```

Осуществляет сравнение двух диапазонов. Возвращает true, если каждый итератор i диапазона [first1,last1) удовлетворяет равенству *i==*(first2+(i-first1)) или предикату comp(*i,*(first2+(i-first1)).

```
InputIterator find(InputIterator first, InputIterator last, const EqualityComparable& value);
InputIterator find_if(InputIterator first, InputIterator last, Predicate pred);
```

Осуществляет поиск и возвращает первый найденный итератор, указывающий на элемент диапазона [first,last), значение которого равно value или удовлетворяет предикату pred.

```
ForwardIterator1 find_end(ForwardIterator1 first1, ForwardIterator1 last1, ForwardIterator2 first2,
ForwardIterator2 last2);
ForwardIterator1 find_end(ForwardIterator1 first1, ForwardIterator1 last1, ForwardIterator2 first2,
ForwardIterator2 last2, BinaryPredicate comp);
```

Осуществляет поиск подпоследовательности в диапазоне [first1,last1), которая идентична [first2,last2). Находит последнюю такую подпоследовательность и возвращает итератор, указывающий на её начало. Сравнение осуществляется с использованием соотношения “равно” или предиката comp.

```
InputIterator find_first_of(InputIterator first1, InputIterator last1, ForwardIterator first2,
ForwardIterator last2);
InputIterator find_first_of(InputIterator first1, InputIterator last1, ForwardIterator first2,
ForwardIterator last2, BinaryPredicate comp);
```

Осуществляет поиск первого вхождения в диапазон [first1,last1) элементов из диапазона [first2,last2). Возвращает первый итератор i диапазона [first1,last1) такой, что для некоторого итератора j диапазона [first2,last2) выполняется равенство *i==*j или предикат comp(*i,*j).

```
UnaryFunction for_each(InputIterator first, InputIterator last, UnaryFunction f);
```

Применяет объект-функцию f к каждому элементу диапазона итераторов [first,last). Возвращает объект-функцию f после того, как он будет применён к каждому элементу.

```
pair<InputIterator1, InputIterator2> mismatch(InputIterator1 first1, InputIterator1 last1,
InputIterator2 first2);
pair<InputIterator1, InputIterator2> mismatch(InputIterator1 first1, InputIterator1 last1,
InputIterator2 first2, BinaryPredicate comp);
```

Осуществляет поиск итератора i диапазона [first1,last1), начиная с которого два диапазона [first1,last1) и [first2,first2+(last1-first1)) отличаются, то есть *i!=*(first2+(i-first1)) или не выполняется предикат comp(*i,*(first2+(i-first1)). Возвращаемое значение представляет собой пару, в которой первый элемент i, а второй элемент *(first2+(i-first1)).

```
ForwardIterator1 search(ForwardIterator1 first1, ForwardIterator1 last1, ForwardIterator2 first2,
ForwardIterator2 last2);
ForwardIterator1 search(ForwardIterator1 first1, ForwardIterator1 last1, ForwardIterator2 first2,
ForwardIterator2 last2, BinaryPredicate comp);
```

Осуществляет поиск в диапазоне [first1,last1) подпоследовательности, в которой каждый элемент идентичен диапазону [first2,last2). Сравнение осуществляется с использованием соотношения “равно” или предиката comp. Возвращает итератор, указывающий на начало найденной подпоследовательности.

```
ForwardIterator search_n(ForwardIterator first, ForwardIterator last, Integer count, const T& value);
ForwardIterator search_n(ForwardIterator first, ForwardIterator last, Integer count, const T& value,
BinaryPredicate comp);
```

Осуществляет поиск подпоследовательности в диапазоне [first,last) из count элементов, равных value или удовлетворяющих предикату comp. Возвращает итератор, указывающий на начало этой подпоследовательности.

Минимум и максимум

```
const T& max(const T& a, const T& b);
const T& max(const T& a, const T& b, BinaryPredicate comp);
```

Возвращает значение наибольшего из двух аргументов. Для сравнения используется соотношение “больше” или предикат comp.

```
ForwardIterator max_element(ForwardIterator first, ForwardIterator last);
ForwardIterator max_element(ForwardIterator first, ForwardIterator last, BinaryPredicate comp);
```

Находит и возвращает итератор на наибольший элемент диапазона [first,last). Для сравнения используется соотношение “ больше ” или предикат comp.

```
const T& min(const T& a, const T& b);
const T& min(const T& a, const T& b, BinaryPredicate comp);
```

Возвращает значение наименьшего из двух аргументов. Для сравнения используется соотношение “меньше” или предикат comp.

```
ForwardIterator min_element(ForwardIterator first, ForwardIterator last);
ForwardIterator min_element(ForwardIterator first, ForwardIterator last, BinaryPredicate comp);
```

Находит и возвращает итератор на наименьший элемент диапазона [first,last). Для сравнения используется соотношение “меньше” или предикат comp.

Обобщенные численные алгоритмы

```
T accumulate(InputIterator first, InputIterator last, T init);
T accumulate(InputIterator first, InputIterator last, T init, BinaryFunction op);
```

Вычисляет и возвращает сумму (или значение, вычисленное с помощью бинарной операции op) всех элементов диапазона [first,last) и значения init.

```
OutputIterator adjacent_difference(InputIterator first, InputIterator last, OutputIterator result);
OutputIterator adjacent_difference(InputIterator first, InputIterator last, OutputIterator result,
BinaryFunction op);
```

Инициализирует элементы диапазона [first1,last1) значением соседних разностей (или результатом выполнения бинарной операции op). Возвращает итератор, указывающий на элемент, следующий за последним вычисленным.

```
T inner_product(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, T init);
T inner_product(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, T init,
BinaryFunction1 op1, BinaryFunction2 op2);
```

Вычисляет и возвращает скалярное произведение элементов диапазона [first1,last1) на элементы диапазона [first2,last2). Начальное значение результата – init. Результат вычисляется выполнением операции result=result+(*i)*(*(first2+(i-first1))) или result=op1(result,op2(*i,*(first2+(i-first1))) с каждым итератором i диапазона [first1,last1).

```
OutputIterator partial_sum(InputIterator first, InputIterator last, OutputIterator result);
OutputIterator partial_sum(InputIterator first, InputIterator last, OutputIterator result, BinaryOperation op);
```

Инициализирует элементы диапазона [first1,last1) значением частной суммы (или результатом выполнения бинарной операции op). Возвращает итератор, указывающий на элемент, следующий за последним вычисленным.

Алгоритмы перестановки

```
void swap(Assignable& a, Assignable& b);
```

Содержимое a присваивается b, а содержимое b присваивается a.

```
void iter_swap(ForwardIterator1 a, ForwardIterator2 b);
```

Эквивалентно swap(*a, *b).

```
ForwardIterator2 swap_ranges(ForwardIterator1 first1, ForwardIterator1 last1, ForwardIterator2 first2);
```

Переставляет каждый элемент диапазона [first1,last1) с соответствующими элементами диапазона [first2,first2+(last1-first1)). Возвращает итератор на элемент, следующий за последним переставленным.

Алгоритмы замены

```
void replace(ForwardIterator first, ForwardIterator last, const T& old_value, const T& new_value);
void replace_if(ForwardIterator first, ForwardIterator last, Predicate pred, const T& new_value);
```

Изменяет значение каждого элемента диапазона [first,last) равного old_value или удовлетворяющего предикату pred на значение new_value.

```
OutputIterator replace_copy(InputIterator first, InputIterator last, OutputIterator result, const T& old_value,
const T& new_value);
OutputIterator replace_copy_if(InputIterator first, InputIterator last, OutputIterator result, Predicate pred,
const T& new_value);
```

Копирует элементы из диапазона [first,last) в диапазон [result, result+(last-first)). При этом вместо элементов имеющих значение old_value копируются элементы имеющие значения new_value. Возвращает итератор на элемент, следующий за последним скопированным.

Алгоритмы удаления

```
ForwardIterator remove(ForwardIterator first, ForwardIterator last, const T& value);
ForwardIterator remove_if(ForwardIterator first, ForwardIterator last, Predicate pred);
```

Удаляет из диапазона [first,last) все элементы, значения которых равны value или удовлетворяют предикату pred. Возвращает итератор на первый элемент диапазона удаляемых элементов.

```
OutputIterator remove_copy(InputIterator first, InputIterator last, OutputIterator result, const T& value);
OutputIterator remove_copy_if(InputIterator first, InputIterator last, OutputIterator result, Predicate pred);
```

Копирует элементы из диапазона [first,last) в диапазон [result,result+(last-first)). При этом элементы, имеющие значение value, не копируются. Возвращает итератор на элемент, следующий за последним скопированным.

Алгоритмы, изменяющие последовательность

```
OutputIterator copy(InputIterator first, InputIterator last, OutputIterator result);
```

Копирует элементы из диапазона [first,last) в диапазон [result,result+(last-first)). Возвращает итератор, указывающий на элемент, следующий за последним скопированным.

```
BidirectionalIterator2 copy_backward(BidirectionalIterator1 first, BidirectionalIterator1 last,
                                     BidirectionalIterator2 result);
```

Копирует элементы из диапазона [first,last) в диапазон [result-(last-first),result). Возвращает итератор, указывающий на элемент стоящий перед последним скопированным.

```
void fill(ForwardIterator first, ForwardIterator last, const T& value);
```

Присваивает значение value каждому элементу диапазона [first, last).

```
OutputIterator fill_n(OutputIterator first, Size n, const T& value);
```

Присваивает значения value каждому элементу диапазона [first,first+n). Возвращает итератор, указывающий на элемент, следующий за последним присвоенным.

```
void generate(ForwardIterator first, ForwardIterator last, Generator gen);
```

Присваивает результат применения функции-объекта gen к каждому итератору диапазона [first,last).

```
OutputIterator generate_n(OutputIterator first, Size n, Generator gen);
```

Присваивает результат применения функции-объекта gen к каждому итератору диапазона [first,first+n). Возвращает итератор, указывающий на элемент, следующий за последним присвоенным.

```
ForwardIterator partition(ForwardIterator first, ForwardIterator last, Predicate pred);
```

Изменяет порядок следования элементов диапазона [first,last) так, что сначала следуют элементы, удовлетворяющие предикату pred, а затем – ему не удовлетворяющие. Возвращает итератор на первый элемент, не удовлетворяющий предикату.

```
void random_shuffle(RandomAccessIterator first, RandomAccessIterator last);
void random_shuffle(RandomAccessIterator first, RandomAccessIterator last, RandomNumberGenerator& rand);
```

Случайным образом изменяет порядок следования элементов диапазона [first,last). Для этого используется стандартные функции генерации случайных чисел или объект-функция rand

```
void reverse(BidirectionalIterator first, BidirectionalIterator last);
```

Изменяет порядок следования элементов диапазона [first,last) на обратный.

```
OutputIterator reverse_copy(BidirectionalIterator first, BidirectionalIterator last, OutputIterator result);
```

Копирует элементы из диапазона [first,last) в диапазон [result,result+(last-first)) так, что последовательность следования элементов становится является обратной. Возвращает итератор, указывающий на элемент, следующий за последним скопированным.

```
ForwardIterator rotate(ForwardIterator first, ForwardIterator middle, ForwardIterator last);
```

Обменивает значения двух диапазонов [first,middle) и [middle,last). Возвращает итератор, указывающий на элемент, следующий за последним обмененным.

```
OutputIterator rotate_copy(ForwardIterator first, ForwardIterator middle, ForwardIterator last,
                           OutputIterator result);
```

Копирует элементы из диапазона [first,last) в диапазон [result,result+(last-first)). При этом сначала помещаются элементы из диапазона [middle,last), а затем из диапазона [first,middle). Возвращает итератор, указывающий на элемент, следующий за последним скопированным.

```
ForwardIterator stable_partition(ForwardIterator first, ForwardIterator last, Predicate pred);
```

Изменяет порядок следования элементов диапазона [first,last) так, что сначала следуют элементы удовлетворяющие предикату pred, а затем – ему не удовлетворяющие. При этом относительный порядок следования элементов сохраняется. Возвращает итератор на первый элемент, не удовлетворяющий предикату.

```
OutputIterator transform(InputIterator first, InputIterator last, OutputIterator result, UnaryFunction op);
```

Изменяет значение элементов, применяя объект-функцию op(*i) к каждому итератору i диапазона [first,last). Результат помещает в последовательность, начинающуюся с итератора result. Возвращает итератор, указывающий на элемент, следующий за последним изменённым.

```
OutputIterator transform(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2,
                         OutputIterator result, BinaryFunction op);
```

Изменяет значение элементов, применяя объект-функцию op(*i1,*i2) к каждому итератору i1 диапазона [first1,last1) и i2 диапазона [first2,last2). Результат помещает в последовательность, начинающуюся с итератора result. Возвращает итератор, указывающий на элемент, следующий за последним изменённым.

```
ForwardIterator unique(ForwardIterator first, ForwardIterator last);
ForwardIterator unique(ForwardIterator first, ForwardIterator last, BinaryPredicate pred);
```

Удаляет все последовательно расположенные элементы с повторяющимися значениями кроме первого. Формирует последовательность удалённых элементов и возвращает итератор на её первый элемент. Сравнение выполняется с использованием соотношения “равно” или предиката pred.

```
OutputIterator unique_copy(InputIterator first, InputIterator last, OutputIterator result);
OutputIterator unique_copy(InputIterator first, InputIterator last, OutputIterator result,
BinaryPredicate pred);
```

Копирует все элементы в последовательность, начинающуюся с итератора result, удаляя последовательно расположенные элементы с повторяющимися значениями кроме первого. Возвращает итератор на элемент следующий за последним скопированным. Сравнение выполняется с использованием соотношения “равно” или предиката pred.

Алгоритмы сортировки

```
void inplace_merge(BidirectionalIterator first, BidirectionalIterator middle, BidirectionalIterator last);
void inplace_merge(BidirectionalIterator first, BidirectionalIterator middle, BidirectionalIterator last,
StrictWeakOrdering comp);
```

Объединяет по месту два отсортированных диапазона [first1,last1) и [first2,last2). Сортировка объединённого диапазона осуществляется в порядке возрастания или порядке, определённом функцией-объектом comp.

```
bool is_sorted(ForwardIterator first, ForwardIterator last);
bool is_sorted(ForwardIterator first, ForwardIterator last, StrictWeakOrdering comp);
```

Возвращает true, если диапазон [first,last) отсортирован в порядке возрастания или порядке определённом объектом-функцией comp.

```
bool lexicographical_compare(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2,
InputIterator2 last2);
bool lexicographical_compare(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2,
InputIterator2 last2, BinaryPredicate comp);
```

Сравнивает два диапазона элементов [first1,last1) и [first2,last2) лексикографически. Для сравнения используется соотношение “меньше” или функцией-объектом comp.

```
OutputIterator merge(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2,
OutputIterator result);
OutputIterator merge(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2,
OutputIterator result, StrictWeakOrdering comp);
```

Объединяет два отсортированных диапазона [first1,last1) и [first2,last2) в один диапазон, начиная с result, отсортированный в порядке возрастания или порядке определённом функцией-объектом comp. Возвращает итератор на элемент объединённого диапазона, следующий за последним.

```
void nth_element(RandomAccessIterator first, RandomAccessIterator nth, RandomAccessIterator last);
void nth_element(RandomAccessIterator first, RandomAccessIterator nth, RandomAccessIterator last,
StrictWeakOrdering comp);
```

Осуществляет разделение диапазона [first,last) таким образом, что до элемента, на который указывает итератор nth, будут находиться элементы равные ему по значению, а после него – все остальные. Сравнение выполняется с использованием соотношения “равно” или функции-объекта comp.

```
void partial_sort(RandomAccessIterator first, RandomAccessIterator middle, RandomAccessIterator last);
void partial_sort(RandomAccessIterator first, RandomAccessIterator middle, RandomAccessIterator last,
StrictWeakOrdering comp);
```

Осуществляет частичную сортировку диапазона [first,last) в порядке возрастания их значений или порядке, определённом объектом-функцией comp. При этом все элементы в диапазоне [first,middle) будут отсортированы, а в диапазоне [middle,last) – нет.

```
RandomAccessIterator partial_sort_copy(InputIterator first, InputIterator last,
RandomAccessIterator result_first, RandomAccessIterator result_last);
RandomAccessIterator partial_sort_copy(InputIterator first, InputIterator last,
RandomAccessIterator result_first, RandomAccessIterator result_last,
Compare comp);
```

Осуществляет частичную сортировку диапазона [first,last) в порядке возрастания их значений или порядке, определённом объектом-функцией comp, и копирует элементы диапазона [first,last) в диапазон [result_first,result_last). Возвращает итератор на элемент, следующий за последним скопированным.

```
void sort(RandomAccessIterator first, RandomAccessIterator last);
void sort(RandomAccessIterator first, RandomAccessIterator last, StrictWeakOrdering comp);
```

Сортирует элементы диапазона [first,last) в порядке возрастания их значений или порядке, определённом объектом-функцией comp.

```
void stable_sort(RandomAccessIterator first, RandomAccessIterator last);
void stable_sort(RandomAccessIterator first, RandomAccessIterator last, StrictWeakOrdering comp);
```

Сортирует элементы диапазона [first,last) в порядке возрастания их значений или порядке, определённом объектом-функцией comp. При этом сохраняет относительный порядок следования элементов с равными значениями.

Алгоритмы для бинарного поиска

```
bool binary_search(ForwardIterator first, ForwardIterator last, const LessThanComparable& value);
bool binary_search(ForwardIterator first, ForwardIterator last, const T& value, StrictWeakOrdering comp);
```

Организует бинарный поиск элемента со значением value в диапазоне [first, last). В процессе поиска диапазон сортируется с использованием соотношения “меньше” или функции-объекта comp. Возвращает true, если элемент со значением value находится в данном диапазоне.

```
pair<ForwardIterator, ForwardIterator> equal_range(ForwardIterator first, ForwardIterator last,
const LessThanComparable& value);
pair<ForwardIterator, ForwardIterator> equal_range(ForwardIterator first, ForwardIterator last, const T& value,
StrictWeakOrdering comp);
```

Организует бинарный поиск элемента со значением value в диапазоне [first, last). В процессе поиска диапазон сортируется с начала и с конца диапазона с использованием соотношения “меньше” или функции-объекта comp. Возвращает пару итераторов [i, j), ограничивающих диапазон, в который элемент со значением value может быть вставлен без нарушения порядка.

```
ForwardIterator lower_bound(ForwardIterator first, ForwardIterator last, const LessThanComparable& value);
ForwardIterator lower_bound(ForwardIterator first, ForwardIterator last, const T& value,
StrictWeakOrdering comp);
```

Организует бинарный поиск элемента со значением value в диапазоне [first, last). В процессе поиска диапазон сортируется с начала с использованием соотношения “меньше” или функции-объекта comp. Возвращает итератор, после которого элемент со значением value может быть вставлен без нарушения порядка.

```
ForwardIterator upper_bound(ForwardIterator first, ForwardIterator last, const LessThanComparable& value);
ForwardIterator upper_bound(ForwardIterator first, ForwardIterator last, const T& value,
StrictWeakOrdering comp);
```

Организует бинарный поиск элемента со значением value в диапазоне [first, last). В процессе поиска диапазон сортируется с конца с использованием соотношения “меньше” или функции-объекта comp. Возвращает итератор, после которого элемент со значением value может быть вставлен без нарушения порядка.