- dans <iterator> :
  - void advance (InputIterator& it, Distance n) :Advances the iterator it by n element positions.
  - distance (InputIterator first, InputIterator last) : Calculates the number of elements between first and last.
  - auto begin(const Container & cont) : Returns an iterator pointing to the first element in the sequence
  - auto end (Container& cont) : Returns an iterator pointing to the past-the-end element in the sequence

- dans <vector> :
  - explicit vector (const allocator_type& alloc = allocator_type()) : Constructs an empty container, with no elements
  - explicit vector (size_type n) : Constructs a container with n elements.
  - vector (size_type n, const value_type& val,
    const allocator_type& alloc = allocator_type()) : Constructs a container with n elements. Each element is a copy of val.
  - vector (initializer_list<value_type> il,
    const allocator_type& alloc = allocator_type()) : Constructs a container with a copy of each of the elements in il, in the same order.
  - vector (InputIterator first, InputIterator last,
    const allocator_type& alloc = allocator_type()) : Constructs a container with as many elements as the range [first,last), with each element emplace-constructed from its corresponding element in that range, in the same order.
  - iterator begin() noexcept : Returns an iterator pointing to the first element in the vector.
  - iterator end() noexcept : Returns an iterator referring to the past-the-end element in the vector container.
  - size_type size() const noexcept : Returns the number of elements in the vector.
  - bool empty() const noexcept : Returns whether the vector is empty (i.e. whether its size is 0).
  - reference operator[] (size_type n) : Returns a reference to the element at position n in the vector container
  - reference at (size_type n) : Returns a reference to the element at position n in the vector. The function automatically checks whether n is within the bounds of valid elements in the vector, throwing an out_of_range exception if it is not (i.e., if n is greater than, or equal to, its size). This is in contrast with member operator[], that does not check against bounds.
  - reference front() : Returns a reference to the first element in the vector.
  - reference back() : Returns a reference to the last element in the vector.
  - void push_back (const value_type& val) : Adds a new element at the end of the vector, after its current last element. The content of val is copied (or moved) to the new element.
  - void pop_back() : Removes the last element in the vector, effectively reducing the container size by one.
  - iterator insert (const_iterator position, const value_type& val) : insert an element before the element at the specified position. The new size is size +1. The return value is the position.
  - iterator erase (const_iterator position) : Removes from the vector a single element (position). The size is reduced by one. The return value is an iterator pointing to the new location of the element that followed the last element era-

sed by the function call. This is the container end if the operation erased the last element in the sequence.

- dans <list> :
  - list() : Constructs an empty container, with no elements.
  - list (initializer_list<value_type> il, const allocator_type& alloc = allocator_type()) : Constructs a container with a copy of each of the elements in il, in the same order.
  - list (InputIterator first, InputIterator last, const allocator_type& alloc = allocator_type()) : Constructs a container with as many elements as the range [first,last), with each element emplace-constructed from its corresponding element in that range, in the same order.
  - begin : Return iterator to beginning
  - end : Return iterator to end
  - empty :Test whether container is empty
  - size : Return size
  - front : Access first element
  - back : Access last element
  - push_front : Insert element at beginning
  - pop_front : Delete first element
  - push_back : Add element at the end
  - pop_back : Delete last element
  - insert : Insert elements
  - erase : Erase elements
  - void sort() : Sorts the elements in the list, altering their position within the container.
  - void remove (const value_type& val) : Removes from the container all the elements that compare equal to val.

- dans <algorithm> :
  - for_each (InputIterator first, InputIterator last, Function fn) : Applies function fn to each of the elements in the range [first,last).
  - InputIterator find (InputIterator first, InputIterator last, const T& val) : Returns an iterator to the first element in the range [first,last) that compares equal to val. If no such element is found, the function returns last.
  - OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result) : Copies the elements in the range [first,last) into the range beginning at result. The function returns an iterator to the end of the destination range (which points to the element following the last element copied).
  - void replace (ForwardIterator first, ForwardIterator last, const T& old_value, const T& new_value) : Assigns new_value to all the elements in the range [first,last) that compare equal to old_value. The function uses operator== to compare the individual elements to old_value.
  - ForwardIterator remove (ForwardIterator first, ForwardIterator last, const T& val) : Transforms the range [first,last) into a range with all the elements that compare equal to val removed, and returns an iterator to the new end of that range. This function returns an iterator to the element that follows the last element not removed.
  - void sort (RandomAccessIterator first, RandomAccessIterator last) : Sorts the elements in the range [first,last) into ascending order.

- map() : Constructs an empty container.
  - map(const map& other) : Copy constructor. Constructs the container with the copy of the contents of other.
  - iterator begin() noexcept : Returns an iterator to the first element of the map.
  - If the map is empty, the returned iterator will be equal to end().
  - iterator end() noexcept : Returns an iterator to the element following the last element of the map. This element acts as a placeholder ; attempting to access it results in undefined behavior.
  - bool empty() const noexcept : Checks if the container has no elements, *i.e.*
  - whether begin() == end().
  - size_type size() const noexcept : Returns the number of elements in the container.
  - T& at(const Key& key) : Returns a reference to the mapped value of the ele- ment with key equivalent to key. If no such element exists, an exception of type std::out_of_range is thrown.
  - T& operator[](Key&& key) : Returns a reference to the value that is mapped to a key equivalent to key, performing an insertion if such key does not already exist.
  - std::pair<iterator, bool> insert(const value_type& value) : Inserts ele- ment into the container, if the container doesn't already contain an element with an equivalent key.
  - iterator erase(iterator pos) : Removes the element at pos.
  - iterator erase(const_iterator first, const_iterator last) : Removes the elements in the range [first; last), which must be a valid range in *this.
  - iterator find(const Key& key) : Finds an element with key equivalent to key.
  - If no such element is found, past-the-end (see end()) iterator is returned.

Dans <unordered_map> :

Unordered map is an associative container that contains key-value pairs with unique keys. Search, insertion, and removal of elements have average constant-time complexity. Internally, the elements are not sorted in any particular order, but organized into buckets. Which bucket an element is placed into depends entirely on the hash of its key. Keys with the same hash code appear in the same bucket. This allows fast access to individual elements, since once the hash is computed, it refers to the exact bucket the element is placed into.

- unordered_map() : Constructs an empty container.
- unordered_map(const unordered_map& other) : Copy constructor. Constructs the container with the copy of the contents of other.
- begin : Returns an iterator to the first element of the map.
- end : Returns an iterator to the element following the last element of the map.
- empty : Checks if the container has no elements.
- size : Returns the number of elements in the container.
- at : Returns a reference to the mapped value of the element.
- [] : Returns a reference to the mapped value of the element.
- insert : Inserts elements into the container.
- erase : Removes elements.
- find : Finds an element.

Dans <set> :

std::set is an associative container that contains a sorted set of unique objects of type Key. Sorting is done using the key comparison function Compare (by default, std::less<Key>).

Search, removal, and insertion operations have logarithmic complexity. Sets are usually implemented as red-black trees.

- set() : Constructs an empty container.
- set(const set& other) : Copy constructor. Constructs the container with the copy of the contents of other.
- iterator begin() noexcept : Returns an iterator to the first element of the set.
    - If the set is empty, the returned iterator will be equal to end().
- iterator end() noexcept : Returns an iterator to the element following the last element of the set. This element acts as a placeholder ; attempting to access it results in undefined behavior.
- bool empty() const noexcept : Checks if the container has no elements, *i.e.*
    - whether begin() == end().
- size_type size() const noexcept : Returns the number of elements in the container.
- std::pair<iterator, bool> insert(const value_type& value) : Inserts ele- ment into the container, if the container doesn't already contain an element with an equivalent key.
- void insert(std::initializer_list<value_type> ilist) : Inserts elements from initializer list ilist. If multiple elements in the range have keys that compare equivalent, it is unspecified which element is inserted.
- iterator erase(iterator pos) : Removes the element at pos.
- iterator erase(const_iterator first, const_iterator last) : Removes the elements in the range [first; last), which must be a valid range in *this.
- iterator find(const Key& key) : Finds an element with key equivalent to key.
    - If no such element is found, past-the-end (see end()) iterator is returned.

Dans <unordered_set> :

Unordered set is an associative container that contains a set of unique objects of type Key. Search, insertion, and removal have average constant-time complexity. Internally, the elements are not sorted in any particular order, but organized into buckets. Which bucket an element is placed into depends entirely on the hash of its value. This allows fast access to individual elements, since once a hash is computed, it refers to the exact bucket the element is placed into.

- unordered_set() : Constructs an empty container.
- unordered_set(const unordered_set& other) : Copy constructor. Constructs the container with the copy of the contents of other.
- begin : Returns an iterator to the first element of the set.
- end : Returns an iterator to the element following the last element of the set.
- empty : Checks if the container has no elements.
- size : Returns the number of elements in the container.
- insert : Inserts elements into the container.
- erase : Removes elements.
- find : Finds an element.