

# Petit résumé de la STL

## 1 Méthodes communes

Tous les conteneurs contiennent les méthodes suivantes :

<code>size_t size()</code>	Renvoie le nombre d'éléments contenus dans le conteneur.
<code>bool empty()</code>	Vérifie si le conteneur est vide.
<code>iterator begin()</code>	Renvoie un itérateur positionné sur le premier élément du conteneur.
<code>iterator end()</code>	Renvoie un itérateur positionné après le dernier élément du conteneur.
<code>iterator rbegin()</code>	Renvoie un itérateur inverse positionné sur le dernier élément du conteneur.
<code>iterator rend()</code>	Renvoie un itérateur inverse positionné avant le premier élément du conteneur.
<code>void insert(pos, elem)</code>	Insère une copie de l'élément à position spécifiée (pour <i>set</i> et <i>map</i> , <i>pos</i> n'est qu'une indication d'un point de départ pour la recherche de la position d'insertion).
<code>erase(pos)</code>	Élimine l'élément se trouvant à la position spécifiée. S'il s'agit d'un conteneur séquentiel, retourne un itérateur positionné sur l'item qui suit celui qui a été retiré. Sinon, ne retourne rien.
<code>erase(deb, fin)</code>	Élimine tous les éléments se trouvant entre les positions <i>deb</i> (inclusivement) et <i>fin</i> (exclusivement). S'il s'agit d'un conteneur séquentiel, retourne un itérateur positionné sur l'item qui suit le dernier qui a été retiré. Sinon, ne retourne rien.
<code>void clear()</code>	Vide le conteneur.

## 2 Interface de `vector<T>`

<code>vector&lt;T&gt;(int n)</code>	Construit un vecteur avec $n$ items initiaux construits par défaut.
<code>vector&lt;T&gt;(int n, T elem)</code>	Construit un vecteur avec $n$ copies de <i>elem</i> .
<code>size_t capacity()</code>	Renvoie le nombre maximal d'éléments que le vecteur peut contenir sans avoir besoin d'une nouvelle réallocation.
<code>void reserve(n)</code>	Fixe la capacité à $n$ éléments, si la capacité actuelle est inférieure (sinon, aucun effet).
<code>T&amp; front()</code>	Renvoie une référence au premier élément.
<code>T&amp; back()</code>	Renvoie une référence au dernier élément.
<code>T&amp; operator[]</code>	Renvoie une référence au $n$ ième élément.
<code>void push_back(elem)</code>	Insère une copie de l'élément à la fin.
<code>void pop_back()</code>	Retire le dernier élément.
<code>size_t size()</code>	Renvoie la taille du vecteur (c'est-à-dire le nombre d'éléments qu'il contient).
<code>void resize(n)</code>	Fixe la taille à $n$ (si la taille augmente, les espaces supplémentaires sont remplis par des objets construits par défaut).

## 3 Interface de `deque<T>`

Le `deque` contient toutes les méthodes de `vector` citées précédemment, sauf `reserve()` et `capacity()`. En plus, il contient les méthodes additionnelles suivantes :

<code>void push_front(elem)</code>	Insère une copie de l'élément au début.
<code>void pop_front()</code>	Retire le premier élément.

## 4 Interface de `list<T>`

<code>list&lt;T&gt;(int n)</code>	Construit une liste avec $n$ items initiaux construits par défaut.
<code>list&lt;T&gt;(int n, T elem)</code>	Construit une liste avec $n$ copies de <i>elem</i> .
<code>T&amp; front()</code>	Renvoie une référence au premier élément.
<code>T&amp; back()</code>	Renvoie une référence au dernier élément.
<code>void push_back(elem)</code>	Insère une copie de l'élément à la fin.
<code>void pop_back()</code>	Retire le dernier élément.
<code>void push_front(elem)</code>	Insère une copie de l'élément au début.
<code>void pop_front()</code>	Retire le premier élément.
<code>void remove(val)</code>	Retire toutes les occurrences d'une valeur.
<code>void remove_if(prédicat)</code>	Retire toutes les occurrences pour lesquelles le prédicat renvoie la valeur <i>true</i> .
<code>void sort()</code>	Trie les éléments de la liste.
<code>void reverse()</code>	Inverse les éléments de la liste.

## 5 Interface commune de `set<T>` et `multiset<T>`

<code>set&lt;T&gt;(const oper)</code>	Constructeur qui reçoit en paramètre l'opérateur de comparaison qui sera utilisé.
<code>int count(elem)</code>	Compte le nombre d'occurrences d'un élément.
<code>iterator find(elem)</code>	Renvoie un itérateur positionné sur la première occurrence de l'élément recherché.
<code>int erase(elem)</code>	Élimine toutes les occurrences d'un élément et renvoie le nombre d'items retirés.

## 6 Méthode unique à `set<T>` et `map<T>`

<code>pair&lt;iterator, bool&gt; insert(elem)</code>	Insère une copie d'un élément. La paire retournée contient la position du nouvel élément et un booléen indiquant si l'insertion s'est faite avec succès, c'est-à-dire si l'élément n'existait pas déjà. <i>Attention</i> : dans le cas d'un <i>map</i> , l'item inséré doit être une paire dont le premier élément est la clé.
--	--

## 7 Méthode unique à `multiset<T>` et `multimap<T>`

<code>iterator insert(elem)</code>	Insère une copie d'un élément et renvoie la position du nouvel élément. <i>Attention</i> : dans le cas d'un <i>multimap</i> , l'item inséré doit être une paire dont le premier élément est la clé.
------------------------------------	---

## 8 Interface commune de `map<T>` et `multimap<T>`

<code>int count(k)</code>	Compte le nombre d'occurrences d'éléments dont la clé est k.
<code>iterator find(k)</code>	Renvoie un itérateur positionné sur le premier élément dont la clé est k.
<code>int erase(k)</code>	Élimine le(s) éléments associé(s) à la clé k et renvoie le nombre d'items retirés.
<code>operator[ ]</code>	Insère une copie d'un élément associé à la clé passée en paramètre.

## 9 Certains algorithmes de la STL

<code>equal(deb1, fin1, deb2)</code>	Vérifie si la séquence allant de <i>deb1</i> à <i>fin1</i> est la même que celle commençant à <i>déb2</i> . Retourne un boolean.
<code>equal(deb1, fin1, deb2, op)</code>	Vérifie si la séquence allant de <i>deb1</i> à <i>fin1</i> est la même que celle commençant à <i>déb2</i> en utilisant une opération de comparaison. Retourne un boolean.
<code>for_each(deb, fin, fonction)</code>	Applique une fonction à tous les éléments compris entre <i>deb</i> (inclusivement) et <i>fin</i> (exclusivement).
<code>count(deb, fin, val)</code>	Compte le nombre d'occurrences d'une valeur.
<code>count_if(deb, fin, prédicat)</code>	Compte le nombre d'éléments pour lesquels le prédicat renvoie <i>true</i> .
<code>min_element(deb, fin)</code>	Renvoie un itérateur sur le plus petit élément.
<code>min_element(deb, fin, oper)</code>	Renvoie un itérateur sur le plus petit élément, en utilisant l'opérateur de comparaison passé en paramètre.
<code>max_element(deb, fin)</code>	Renvoie un itérateur sur le plus grand élément.
<code>max_element(deb, fin, oper)</code>	Renvoie un itérateur sur le plus grand élément, en utilisant l'opérateur de comparaison passé en paramètre.
<code>generate(deb, fin, f)</code>	Remplace tous les items de la séquence spécifiée par la valeur obtenue en appliquant la fonction <i>f</i> (qui ne prend aucun argument).
<code>generate_n(pos, n, f)</code>	À partir de la position indiquée par l'itérateur <i>pos</i> , remplace <i>n</i> items par la valeur obtenue en appliquant la fonction <i>f</i> (qui ne prend aucun argument).
<code>fill(deb, fin, val)</code>	Remplace par la valeur <i>val</i> tous les items de la séquence spécifiée.
<code>fill_n(pos, n, val)</code>	À partir de la position indiquée par l'itérateur <i>pos</i> , remplace <i>n</i> items par la valeur spécifiée.
<code>find(deb, fin, val)</code>	Renvoie un itérateur qui pointe sur la première occurrence de la valeur cherchée.
<code>find_if(deb, fin, prédicat)</code>	Renvoie un itérateur qui pointe sur le premier élément pour lequel le prédicat renvoie <i>true</i> .
<code>copy(deb1, fin1, deb2)</code>	Copie tous les éléments de <i>deb1</i> (inclusivement) à <i>fin1</i> (exclusivement) dans un autre conteneur, à partir de la position <i>deb2</i> .
<code>transform(deb1, fin1, deb2, fonct)</code>	Applique la fonction à tous les éléments de <i>deb1</i> à <i>fin1</i> et, pour chaque élément, met le résultat de la fonction dans un autre conteneur, à partir de la position <i>deb2</i> .

<code>replace(deb1, fin1, val1, val2)</code>	Remplace toutes les occurrences de <i>val1</i> par la valeur <i>val2</i> .
<code>replace_if(deb, fin, prédicat, val)</code>	Remplace par <i>val</i> tous les éléments pour lesquels le prédicat renvoie <i>true</i> .
<code>remove(deb, fin, valeur)</code>	Retire toutes les occurrences qui sont égales à la valeur passée.
<code>remove_if(deb, fin, prédicat)</code>	Retire toutes les items pour lesquels le prédicat renvoie <i>true</i> (fonctionne seulement avec les conteneurs séquentiels).
<code>reverse(deb, fin)</code>	Inverse l'ordre des éléments du conteneur (seulement pour les conteneurs séquentiels).
<code>search(deb1, fin1, deb2, fin2)</code>	Cherche entre <i>deb1</i> et <i>fin1</i> du premier conteneur, la sous séquence entre <i>deb2</i> et <i>fin2</i> du deuxième conteneur. Retourne un itérateur sur le début de la séquence du premier conteneur.
<code>sort(deb, fin)</code>	Trie les éléments du conteneur (seulement pour les conteneurs séquentiels).

## 10 Introduction

<code>back_inserter(conteneur)</code>	Ajoute un élément à la fin du conteneur.
<code>front_inserter(conteneur)</code>	Ajoute un élément au début du conteneur.
<code>inserter(cont, cont.begin())</code>	Ajoute un élément pour un conteneur associatif.

## 11 Foncteurs prédéfinis de la STL

<code>negate&lt;T&gt;( )</code>	<code>- param</code>
<code>plus&lt;T&gt;( )</code>	<code>param1 + param2</code>
<code>minus&lt;T&gt;( )</code>	<code>param1 - param2</code>
<code>multiplies&lt;T&gt;( )</code>	<code>param1 * param2</code>
<code>divides&lt;T&gt;( )</code>	<code>param1 / param2</code>
<code>modulus&lt;T&gt;( )</code>	<code>param1 % param2</code>
<code>equal_to&lt;T&gt;( )</code>	<code>param1 == param2</code>
<code>not_equal_to&lt;T&gt;( )</code>	<code>param1 != param2</code>
<code>less&lt;T&gt;( )</code>	<code>param1 &lt; param2</code>
<code>greater&lt;T&gt;( )</code>	<code>param1 &gt; param2</code>
<code>less_equal&lt;T&gt;( )</code>	<code>param1 ≤ param2</code>
<code>greater_equal&lt;T&gt;( )</code>	<code>param1 ≥ param2</code>
<code>logical_not&lt;T&gt;( )</code>	<code>!param</code>
<code>logical_and&lt;T&gt;( )</code>	<code>param1 &amp;&amp; param2</code>
<code>logical_or&lt;T&gt;( )</code>	<code>param1    param2</code>
<code>bind2nd(op, valeur)</code>	Objet fonction prédéfini qui prend une fonction à deux arguments et la transforme en une nouvelle fonction à un seul argument où le second argument est la valeur spécifiée.