Points clés sur les conteneurs en C++

Michel Simatic et Amina Guermouche

Télécom SudParis

7 avril 2025

Plan

1. List
2. Map et compagnie (multimap, unordered map)
3. Que choisir ?
4. Quelques trucs à connaître

List

1. → List
2. Map et compagnie (multimap, unordered map)
3. Que choisir ?
4. Quelques trucs à connaître

Classe `List`

- Définition

```
std::list<int> liste1;                  // Definition d'une liste vide
std::list<int> liste2{ 7, 5, 16, 8 }; // Definition d'une liste de 4 elements
```

- Quelques fonctions utiles de la classe list :
  - `push_front` : ajout en tête
  - `push_back` : ajout en queue
  - `pop_front` : supprime le premier élément
  - `pop_back` : supprime le dernier élément
  - `insert(iterator_position, valeur)`
- Parcours : *iterator* ou *range based loop* comme pour les vecteurs.

Différences entre vecteurs et listes

Opération	Vecteur	List	Remarque
Parcours	O(n)	O(n)	
Ajout fin	O(n)	O(1)	Vecteur $\rightarrow$ O(1) s'il y a de la place
Ajout	O(n)	O(1)	Pour le vecteur, il faut copier les données de son tableau
Suppression à la fin	O(1)	O(1)	
Suppression ailleurs	O(n)	O(1)	Pour le vecteur, il faut déplacer les données de son tableau
Recherche pas triée	O(n)	O(n)	
Recherche triée	O(log n)	O(n)	Avec la liste, on n'a pas moyen d'accéder à une "case" particulière

Map et compagnie (multimap, unordered map)

1. List
2. → Map et compagnie (multimap, unordered map)
3. Que choisir ?
4. Quelques trucs à connaître

Classe `Map`

- Définition : Une map est une séquence ordonnée de couples (clé, valeur) ((*key*, *value*)), le premier (respectivement second) élément s'appelant *first* (respectivement *second*).

```
std::map<char, int> m1;
std::map<char, int> m2{ {'a', 1}, {'b', 2} };
---

- Parcours : _iterator_ ou _range based loop_ comme pour les vecteurs.

```cpp
for(const auto & i : m)
  std::cout << i.first << "," << i.second << std::endl;
// Depuis C++17
for (const auto& [key, value] : m)
  std::cout << key << "," << value << std::endl;
```

- NB : Il est possible d'accéder aux éléments en utilisant `m[cle]`

```
value = m[cle];
m[cle] = value;
```
```

Quelques remarques sur les map

- Une map est triée selon l'ordre de clé (ordre croissant par défaut ou ordre défini par l'utilisateur·trice).
- Dans la STL, une map est implémentée en arbre binaire de recherche.
- Lors de l'insertion d'un élément, la map vérifie si un élément avec la même clé existe déjà. Si c'est le cas, le nouvel élément n'est pas inséré.
- Étant donné que le tri par défaut dans une map utilise l'opérateur `<` (*strict weak order*), il est nécessaire de le surcharger pour des types dé?nis par l'utilisateur. Sinon, il y aura des erreurs de compilation lors de l'insertion d'éléments.

Surcharge de l'opérateur `<`

- Condition du *strict weak order*
  - $a < a \rightarrow false$
  - $(a < b) \wedge b < c \rightarrow a < c$
  - $!(a < b) \wedge !(b < a) \rightarrow a \equiv b$
- Exemple tiré du bridge (où la valeur de la carte importe, mais sa couleur -trèfle, carreau, cœur, pique- aussi)

```
bool operator<(const card &a, const card &b)
{
  return a.couleur < b.couleur ||
         (a.couleur == b.couleur && a.valeur < b.valeur);
}
```

Classe `multimap`

- Définition : Une multimap est une map dans laquelle plusieurs valeurs peuvent avoir la même clé.
- Une multimap est triée selon la fonction de comparaison des clés.
- Deux éléments avec une même clé seront triés selon l'ordre d'insertion.
- Une multimap est également représentée sous forme d'arbre binaire de recherche.

Classe `unordered map` (≡ classe Java `HashMap`)

- Une *unordered map* est également une séquence de couple <clé, valeur>.
- Une *unordered map* est représentée par une table de hashage dont les clés sont transformées en indices de la table grâce à la fonction de hashage.
- Les clés ne sont pas triées.

Différences entre une map et une unordered map

Opération	Map	Unordered map	Remarque
Recherche	O(log(n))	O(1)	Pour unordered map, O(n) si toutes les valeurs sont sur le même hash
Ajout	O(log(n))	O(1)	Idem
Parcours	Trié	Non trié	

Que choisir ?

1. List
2. Map et compagnie (multimap, unordered map)
3. $\rightarrow$ Que choisir ?
4. Quelques trucs à connaître

Quel conteneur choisir ?

- Utiliser un vecteur sauf si vous avez une bonne raison de ne pas le faire.
- Utiliser une map quand vous voulez faire une recherche basée sur une valeur.
- Utiliser une unordered map si :
  - La clé est intrinsèquement sans ordre.
  - OU BIEN vos tests de performance révèlent des problèmes qu'un hashage pourrait résoudre (cf. page 444 de "C++ Primer" 5th edition, 2012)

Quelques trucs à connaître

1. List
2. Map et compagnie (multimap, unordered map)
3. Que choisir ?
4. $\rightarrow$ Quelques trucs à connaître

Transformation d'un conteneur vers un autre

- Boucle classique avec `push_back`
- Utilisation d'itérateur

```
map<int, int> m;
// On remplit la map
vector<pair<int, int>> v{m.begin(), m.end()};
```

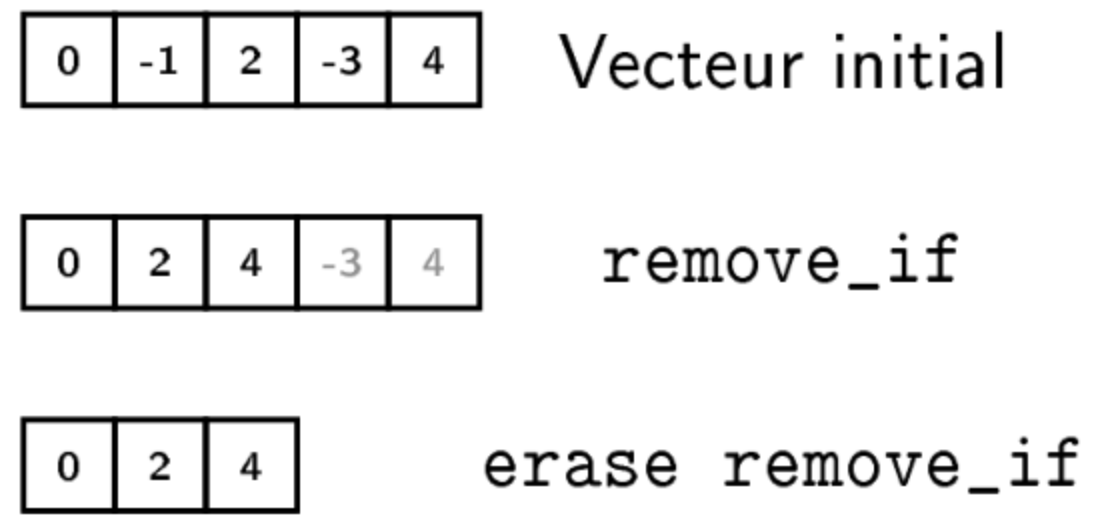
Suppression d'un élément de liste (1/2)

- `remove(iterator_begin, iterator_fin, valeur)`
- `remove_if(iterator_begin, iterator_fin, valeur, unary_predicate_returning_bool)`
- **NB** : Ces fonctions ne suppriment pas l'espace alloués aux éléments supprimés. Les éléments sont cependant inaccessibles. Ce comportement est dû au fait que ces fonctions ne sont pas uniquement appliquées aux conteneurs.

Suppression d'un élément de liste (2/2)

- Combiner `erase` et `remove` (ou `remove_if`) pour supprimer les éléments ⇒ Idiome *erase remove*.
  - Exemple :

```cpp
v.erase(std::remove_if(v.begin(), v.end(), [](const int &e){return e < 0;}), v.end()); // supprime tous les elements negatifs
```



- En C++20, préférez `erase(iterator_begin, iterator_fin, valeur)` et `erase_if` qui évitent ces deux étapes (cf. cours sur les vecteurs).

Questions ?

1. List
2. Map et compagnie (multimap, unordered map)
3. Que choisir ?
4. Quelques trucs à connaître