Les conteneurs standards

Imaginons un instant que nous devions gérer une bibliothèque de livres sur le C++. Nous en avons beaucoup et nous voudrions en ajouter de nouveaux, en supprimer d'autres, en bref, faire comme dans la réalité. Pour commencer, résumons un livre par son nom et supposons que nous en avons 4. Le code suivant vous est peut-être venu à l'esprit.

```
string const premier_livre { "The C++ language" };
string const deuxieme_livre { "C++ Primer" };
string const troisieme_livre { "C++ concurrency in action" };
string const quatrieme_livre { "Qt 5 les essentiels" };
```

Maintenant, j'ai non plus 4 mais 15 livres. Va t-on vraiment créer 15 variables ? Et si j'ai 100 livres ? Non, ce n'est vraiment pas une bonne solution. En plus, le type ne change pas ; n'y aurait-il pas un moyen de ne pas à avoir à réécrire le type (+ d'éventuels attributs) X fois ? Heureusement, la bibliothèque standard nous fournit des outils formidables : les conteneurs.

Qu'est ce qu'un conteneur

Un conteneur est un outil fourni par la bibliothèque standard qui permet de regrouper autant d'éléments dénombrables de même type que l'on veut ; dans notre exemple, les livres sont bien de même type (string const) et son bien dénombrables. On parle de **collection** d'éléments. Et en plus de regrouper des informations de même type, un conteneur fournit les moyens de les modifier, d'en ajouter, d'en supprimer, de les compter, etc. Examinons tout cela.

Des conteneurs standards

La bibliothèque standard nous fournit plusieurs conteneurs, mais deux vont particulièrement retenir notre attention :

- array : permet de stocker un nombre fixe d'éléments ;
- vector : permet de stocker un nombre variable d'éléments ;

Il en existe plusieurs autres, mais ces deux-là sont parmi les plus utilisés et sont simples à comprendre. Et pour les utiliser, il faut commencer par les inclure avec les instructions suivantes (respectivement) :

- #include <array>
- #include <vector>

Initialisation

Commençons par std::array. Comme déjà dit, ce conteneur est une collection à taille fixe d'éléments. Il prend comme paramètres un type et une taille.

```
[const] std::array<type, taille> identificateur;
```

Quant à vector, sa taille est variable, il ne faut donc que le type.

```
[const] std::vector<type> identificateur;
```

Notez que, pour les deux conteneurs, il est possible de préciser l'attribut const ou non ; tout dépend de si la collection peut être modifiée ou non. Notez bien cependant que dans le cas de vector, s'il est déclaré avec const, alors il est impossible par la suite de rajouter ou de supprimer des éléments.

Si l'on reprend notre exemple de la bibliothèque, notre collection pourrait se faire ainsi :

```
std::array<std::string, 4> bibliotheque_array; // Si notre bibliothèque ne contient que 4 livres et uniquement 4
std::vector<std::string> bibliotheque_vector; // Si l'on veut ajouter ou supprimer des livres plus tard
```

Maintenant, passons à l'initialisation à proprement parler. C'est exactement la même syntaxe que celle vue précédemment pour les variables :

```
std::array<std::string, 4> bibliotheque_array {"The C++ language", "C++ Primer", "C++ concurrency in action", "Qt 5 les essentiels"};

std::vector<std::string> bibliotheque_vector {"The C++ language", "C++ Primer", "C++ concurrency in action", "Qt 5 les essentiels"};
```

Parcourir un conteneur

Pour l'instant, c'est bien beau, mais nous sommes quand même assez limités. Heureusement, les conteneurs contiennent tout un panel de fonctions qui vont nous être bien utiles. Mais avant de les examiner en détails, abordons une notion essentielle pour la suite du cours.

Les itérateurs

Qu'est ce qu'un itérateur ? Les itérateurs sont une abstraction fournie par les conteneurs pour les parcourir et les manipuler. On dispose ainsi d'une méthode générique car commune à tous les conteneurs. Mais comment marchent-ils ? C'est ce que je vous propose de découvrir à travers quelques exemples.

```
#include <iostream>
#include <vector>
```

```
int main()
{
    std::vector<int> v {2, 7, 42, -451, 5404, 0};

    for (auto it = v.begin(); it != v.end(); ++it)
    {
        std::cout << "Valeur courante : " << *it << std::
endl;
    }

    return 0;
}
```

Essayez de compiler ce code et vous verrez qu'il affiche toutes les valeurs de notre vector. Expliquons ce code plus en détails.

La première ligne est simple à comprendre, nous déclarons un ``vector`` de ``int`` et l'initialisons avec des valeurs bidons. C'est le ``for`` qui est vraiment intéressant.

```
auto it = v.begin()
```

Même type, plusieurs éléments, dénombrable.

Un peu lourd à écrire, voir impossible si beaucoup d'élément

Définition conteneurs : collection d'éléments de même type. Dénombrable, notion de premier élément, de dernier élément, d'élément suivant, de nombre d'éléments.

Initialisation : avec initilizer-list

Fonction begin, end, next et size (prev, advance, distance)

Modification : push_back (emplace ? emplace_back ? insert ? pop_back ?)

Déréférencement * + cout/variable

code d'exemple avec array et vector. Différence compile time et runtime.

Existe d'autres conteneur, sera abordé plus tard et possible de créer ses

propres conteneur (chapitre structure de données). Le plus important est de respecter la sémantique (ie la syntaxe permettant de les manipuler)

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++