

Les éléments d'une collection

Possibilité de manipuler directement les éléments d'une collection. Via itérateur (le plus générique) et pour certains types de collection par [] ou at().

Déréferencer un itérateur

Lorsque un itérateur correspond à une position valide dans une collection (donc pas end), il est possible d'accéder à l'élément en utilisant l'opérateur * devant l'itérateur :

```
vector<int> v { 1, 2, 3, 4, 5 };  
auto p = begin(v);  
cout << (*p) << endl;
```

affiche 1

Egalement accessible en modification (si non const) :

```
auto p = begin(v) + 3;  
*p = 0;  
for (auto value: v)  
    cout << value << endl;
```

Si fonction template :

```
template<class Iterator>  
void foo(Iterator it) {  
    auto value = *it;  
}
```

it est un itérateur sur un élément d'un conteneur, donc *it correspond à un élément d'un conteneur. Le type déduit par auto est le type de cet élément

```
std::vector<int> v {};  
foo(begin(v)); // dans foo, it est de type  
vector<int>::iterator et *it est de type int
```

Il est possible d'expliciter auto. Pour cela, il faut pouvoir récupérer le type d'un itérateur et le type de valeur pointé. Pour cela, il faut utiliser la classe de trait `iterator_traits` :

```
template<class Iterator>  
void foo(Iterator it) {  
    typename iterator_traits<Iterator>::value_type value = *  
it;  
}
```

[explication sur typename ? sur classe de traits ?](#)

On comprend l'intérêt de auto dans ce code.

Se déplacer

next, prev, advance, distance

Accès aléatoire aux éléments

Vu comment accéder séquentiellement dans un conteneur (du début à la fin). Méthode la plus simple et la plus performante. Autre méthode : accéder directement à un élément dans le conteneur. Faisable sur certain type de conteneur, dont array et vector.

Utilisation de `at()` et `[]`

Vérification des limites d'accès

Chapitre précédent	Sommaire principal	Chapitre suivant
------------------------------------	------------------------------------	----------------------------------

[Cours, C++](#)