Un outil indispensable pour le C++ moderne

christophe.labourdette(at)cmla.ens-cachan.fr

Octobre 2016

Standard Template Library

La STL est une librairie générique permettant de manipuler des collections de données à l'aide d'algorithmes modernes et efficaces.
Elle permet en ajoutant un niveau d'abstraction de pratiquer une programmation générique.
Elle comprend trois composants importants :

- Conteneurs
- Itérateurs
- Algorithmes

Historique

- Issue des travaux de Stepanov sur la programmation générique (1979),
- En 1993 le comité ANSI/ISO lui demande un avant-projet,
- En 1994 il est amendé et validé,
- HP décide en 1994 de rendre public les sources développés par Stepanov, Lee et Musser,
- Ces sources sont à la base de toutes les implémentations actuelles.

le namespace std

La librairie standard est construite dans le namespace std.
Il y a de fait 3 utilisations possibles :

1. directement :

```
std::cout << ''coucou'' << std::endl;
```

2. à l'aide de déclarations using

```
using std::cout;
using std::endl;
cout << ''coucou'' << endl;
```

3. avec une directive using

```
using namespace std;
cout << ''coucou'' << endl;
```

conteneurs

Les conteneurs sont utilisés pour stocker les collections d'objets d'un certain type. Chaque type de conteneur a ses avantages et ses désavantages.

- Conteneurs ordonnés :
  array, vector, deque, list
- Conteneurs associatifs :
  set, multiset, map, multimap
- Conteneurs non-ordonnés :
  unordered set, multiset, map, multimap

Conteneurs ordonnés

- **vector** : il s'agit d'un tableau dynamique on peut ajouter et supprimer des éléments à la fin,
- **array** : c'est un tableau statique,
- **deque** : double-ended queue c'est donc une sorte de tableau dynamique qui peut grossir dans les deux directions,
- **list** : une liste chaînée classique on peut insérer ou supprimer un élément n'importe où rapidement,

Conteneurs associatifs

- **set** : une collection dans laquelle chaque élément est trié selon sa propre valeur. On ne peut avoir de doublon.
- **multiset** : la même chose qu'un **set** mais on peut avoir des doublons.
- **map** : contient des des éléments qui sont des paires de (clés, valeurs), on ne peut pas avoir de doublon dans les clés.
- **multimap** : la même chose que **map** mais on peut avoir des doublons.

conteneurs non-ordonnés

Dans un conteneur non-ordonné les éléments n'ont pas d'ordre, la seule chose qui compte c'est l'appartenance ou non au conteneur.

- **unordered set** : Un ensemble, chaque élément est unique.
- **unordered multiset** : la même chose qu'un **unordered set** mais on peut avoir des doublons.
- **unordered map** : contient des des éléments qui sont des paires de (clés, valeurs), on ne peut pas avoir de doublon dans les clés.
- **unordered multimap** : la même chose que **unordered map** mais on peut avoir des doublons.

Iterateur

Un iterateur est un objet qui peut parcourir les collections. Un itérateur représente une certaine position dans le conteneur. Les opérations suivantes définissent le comportement d'un itérateur :

- **Operator *** renvoie l'élément de la position courante
- **Operator ++** passer à l'élément suivant
- **Operator ==** ou **!=** testent si deux opérateurs représentent la même position
- **Operator =** assigne un opérateur

l'interface est identique à celle des pointeurs
begin() et **end()** sont des fonctions qui renvoient le début et la fin du conteneur.

Algorithmes

La STL fournit des algorithmes pour manipuler les éléments des collections.
Ces algorithmes offrent en général des services tels que : chercher, trier, copier, réordonner, modifier ou faire des calculs.
Les algorithmes ne sont pas des fonctions membres des conteneurs mais des fonctions globales qui agissent utilisent des itérateurs.

vector

```
#include <vector>
#include <iostream>
...
vector<int> v1;

for (int i=0;i<12;i++)
 v1.push_back(i);
for (int i=0;i< v1.size();i++)
  std::cout << v1[i} << ' ';
std::cout << std::endl;
```

list

```
#include <list>
#include <iostream>
...
list<char> liste;
for (char c = 'a'; c <= 'z'; c++)
  liste.push_back(c);
for (char elt : liste)
  std::cout << elt << ' ';
std::cout << std::endl;
```

map

```
#include <map>
#include <string>
#include <iostream>
...
  std::map<int,std::string> collection;
  collection = { {1,"bruno"},
 {2,"nadine"},
 {3,"jules"}};
  for (auto a : collection)
    std::cout << a.second << " ";
  std::cout << std::endl;
  collection[2]="cesar";
  std::cout << collection[2] << std::endl;
```

Auto

Le C++11 a apporté de nombreuses nouveautés et le mot clé auto est l'une des nouvelles fonctionnalité.
Il permet de déclarer des variables en déduisant leur type automatiquement.

```
auto i=45; // i est de type int
double g();
auto x=g(); // x est de type double
```

Attention le mécanisme de déduction peut être complexe et pas nécessairement facile à bien utiliser.

Nouveau for

Le C++11 apporte une nouvelle boucle for très utile pour parcourir des collections.
Lorsqu'on le couple à l'utilisation de auto on peut par exemple :

```
std::vector<float> vecteur;
...
for (auto& y : vecteur) {
  y *= 0.3;
}
```

la variable y parcourt la collection vecteur et peut également modifier celle-ci grâce à l'utilisation d'une référence.

for vs iterateur

```
for (auto elt : coll) {
      ...
}
for (auto p=coll.cbegin();p!=coll.cend();++p)
{
    ...
  auto elem = *p;
    ...
}
```

On dispose pour les containers de
begin() et end() du container :: iterator et
cbegin() et cend() du container :: const_iterator

fonctions lambdas

C++11 fournit des fonctions lambdas, ce sont des objets fonction anonymes ”inline” dans le code, pouvant être placés partout, y compris en tant qu'argument d'une fonction par exemple.
La version la plus simple est :

```
auto p = [] () { std:cout << "Je suis une fonction
  << std::endl; };
p(); // Appel de la fonction
```

Il est possible de spécifier des passages par valeur ou par référence :

[&] : passage par adresse
[=] : passage par copie
[&aa,=] : 1er paramètre passé par adresse, second par copie

lambdas (suite)

On peut également préciser le type de retour avec la syntaxe suivante : [] () -> int { return a*a; };
Voici un exemple de tri :

```
#include <vector>
#include <cmath>
#include <iostream>
#include <algorithm>
   ...
std::vector<int> v={50,-10,20,-30};

std::sort( v.begin(), v.end(),
     [](int a, int b) { return abs(a)<abs(b); });
for( int t = 0; t<v.size(); t++ )
    std::cout << v[t] << " ";
std::cout << std::endl;
```