Programmation avancée en C++: STL 2

Emmanuel Franck

Bureau 225, UFR math info
mail: emmanuel.franck@inria.fr

Conteneurs associatif

- Jusqu'a présent on a étudié les conteneurs sequentiels qui correspondent a des éléments ordonnés que l'on parcours avec un itérateur.
- **Exemple**: vector, list, dequeue etc.
- Les conteneurs associatifs n'utilise pas des données ordonnées, mais par un système de clé.
- Exemple:
  - un carnet d'adresse: on retrouve le numéro, l'adresse (les données) à partir du nom (la clé),
  - L'apprentissage par renforcement (a voir).

1. Conteneurs associatif: map

Conteneur ”map”

- Le conteneur ”map” est un conteneur qui permet d'associer une ”clé” à une donnée.
- **Exemple**: un nom avec un numéro de téléphone.
- Cela correspond au dictionnaire en Python.

Important

Les conteneurs ”map” impose d'unicité des clés. Une clé = une valeur associée. Pas plus.

Important

Les conteneurs ”map” nécessite d'ordonner les clés. Par conséquent il faut avoir une relation d'ordre (ici $<$) sur le type des clés.

- Connaissant une clé l'accès à la valeur associée est possible en $O(\log(N))$ avec N le nombre de clés.

Conteneur ”map”: premier exemple

- On va se donner des clés de type ”char” et des données de type ”int”.

```
1 #include <map>
2 ...
3 map<chat,int> first_map;
4 cout<<"taille du map >> "<<first_map.size()<<endl;
```

```
user $ taille du map >> 0
```

- ”Size” permet de déterminer le nombre de couple (clé, donnée) dans le conteneur.
- Comment insérer un élement ?

```
1 first_map['S']=200
2 cout<<"taille du map >> "<<first_map.size()<<endl;
```

```
user $ taille du map >> 1
```

Conteneur ”map”: premier exemple II

- Contrairement au vecteur ou le crochet permet d'accéder à un élément existant, ici il permet d'insérer un élément.

```
1 cout<<first_map['M']<<endl;
2 cout<<"taille du map >> "<<first_map.size()<<endl;
3 cout<<first_map['S']<<endl;
4 cout<<"taille du map >> "<<first_map.size()<<endl;
```

```
user $ 0
user $ taille du map >> 2
user $ 200
user $ taille du map >> 2
```

- Il a crée le couple ('M',0). Le crochet crée une donnée si elle existe pas, y accède si elle existe déjà.

Remarque

”Map” est une template de classe.

Accès aux données d'un "map"

- On souhaite parcourir les données d"un conteneur "map". Comment faire ?

```
1  map<char,int> :: iterator im;
2     for (im=first_map.begin(); im!=first_map.end(); im++){
3         cout << (*im).first <<" " << (*im).second<<endl;
4     }
```

ou

```
1     for (auto im=first_map.begin(); im!=first_map.end(); im++){
2         cout << (*im).first <<" " << (*im).second<<endl;
3     }
```

```
user $ M 0
user $ S 200
```

- "First" donne la clé et "second" la donnée associée.

Accès aux données d'un "map" et constructeur

- Depuis le C++11, on peut faire comme en python.
- Boucle sur les éléments d'un conteneur directement:

```
1 for (auto el : first_map){
2      cout << el.first <<" " << el.second<<endl;
3  }
```

- Comment on construit un conteneur "map"

```
1 map<char,double> m; // construction, vide
2 map<char,double> m2 (m); // construction direct à partir d'un autre
3 map<char,double> m2 {m}; // construction direct à
  partir d'un autre en c++11
4 map<char,double> m2 =m; // construction avec copie
5 map<char,double> m2 ={m}; // construction avec copie en c++11
```

- les clés sont ordonnées pas défaut avec <. Pn peut le faire aussi avec >

```
1 map< char,double, greater<char> > m;
2 map< int,double, less<int> > m;
```

Notion de paire I

- On voit que finalement un ”map” va prendre comme données une **paire**: clé donnée.
- Afin de faciliter l'utilisation des map il existe la classe ”pair”.

```
1 paire<char,double> m('c',1.32); // constructeur
2 auto p = paire<char,double> ('c',1.32); // affectation
3 auto p = make_pair('c',1.32)
4 cout << p.first<<" " <<p.second<<endl;
```

- ”make_pair” estime le type de chaque élément de la paire en fonction des valeurs (comme auto).
- La classe dispo de deux opérateurs: ”==” et ”¡”.
- $<$ compare d'abord la clé puis la valeur associée.

Notion de paire II

- Le conteneur ”map” contient un constructeur qui prend en argument:

```
1 initializer_list<pair<T,U>>
```

- On se souvient que ”initializer_list” permet d’avoir une de taille non déterminer de paramètre.
- On voit que ca utilise les templates pour que l’outil soit général.
- Utilisation:

```
1 map<char,double> m { {'c',10}, {'e',12}, {'f',13} };
2 pair<char,double> p ={'x',3};
3 map<char,double> m2 { p, {'f',13} };
```

Relation d'ordre pour map

- Les clés sont ordonnées pas défaut avec <. On peut le faire aussi avec >.

```
1 map< char,double, greater<char> > m;
2 map< int,double, less<int> > m;
```

- On peut imaginer des cas ou la relation d'ordre utilisée est pas vraiment connu.
- On souhaite la récupérer pour la tester.

```
1  auto comp_func = m.key_comp();
2  auto comp_value = m.value_comp()
```

- Le résultat obtenu est une fonction directement utilisable dans des if etc.
- Le choix de la relation d'ordre influe sur le résultat.

Relation d'ordre pour map II

Important

Dans un "map" les clés sont uniques, il ajoute un élement que si la clé n'existe pas. Il faut donc choisir une bonne relation d'ordre "¡".

Important

L'égalité entre deux clés est déduite par $x < y$ et $y < x$.

Exemple I

- Bonne relation d'ordre

```
1  class point{
2      public:
3      double x;
4      double y;
5      point(): x(0.0), y(0.0){};
6      point(double a,double b): x(a), y(b){};
7      bool operator <(const point & p) const {
8          auto norm2=x*x+y*y;
9          auto norm2_2=p.x*p.x+p.y*p.y;
10         return (norm2 < norm2_2) ;}
11 };
12 int main(){
13     point p1(1.0,1.0), p2(1.0,2.0);
14     map<point,double> ma;
15     ma[p1]=1.2, ma[p2]=1.3;
16     for (auto el : ma){
17      cout << el.first.x << " "<< el.first.y <<" " << el.second<<endl;}
18 }
```

```
user $ 1 1 1.2
user $ 1 2 1.3
```

Exemple II

- Mauvaise relation d'ordre

```
1  class point{
2      public:
3      double x;
4      double y;
5      point(): x(0.0), y(0.0){};
6      point(double a,double b): x(a), y(b){};
7      bool operator <(const point & p) const {return (x < p.x) ;}
8  };
9  int main(){
10     point p1(1.0,1.0), p2(1.0,2.0);
11     map<point,double> ma;
12     ma[p1]=1.2, ma[p2]=1.3;
13     for (auto el : ma){
14      cout << el.first.x << " "<< el.first.y <<" " << el.second<<endl;}
15 }
```

```
user $ 1 2 1.3
```

- Il a écrasé la valeur précédente. Car il a considéré que les deux étaients égaux.

Accès aux élements

- Comme vu précédemment, on peut accéder à un élément avec le crochet:

```
1 map<char,int> m;
2 m['S']=2
3 cout<< m['S']<<endl;
```

- Pour *map* $< U, T >$: si la clés existent pas, on crée l'élement make_pair(key,T()).
- Ca marche aussi avec les "iterator":

```
1 map<char,int> m;
2 map<char,int> :: iterator il;
3 *il = make_pair("S",1);
4 cout<<*il<<endl;
```

Accès aux élements

- ’Find” permet de trouver toutes les paires associées a une clé:

```cpp
#include <iostream>

using namespace std;
#include <map>
int main()
{
    map <char,double> m;
    map <char,double> :: iterator im;
    m['s']=1.2;
    im = m.find('s');
    cout<< (*im).second<<endl;
    return 0;
}
```

```
user $ 1.2
```

2. Conteneurs associatif: multimap et set

Conteneur ”multimap”

- Contrairement a ”map”, on peut avoir plusieurs fois la la même clé ou des clés équivalentes.

Important

Dans ce cadre l’opérareur ”[]” n’as plus vraiment de sens.

- ”Erase” lui peut supprimer plusieurs élements contrairement a ”map.
- ”count(”key”) donne le nombre d’élements associés à cette key.

```
1    cout<<lower_bound('c')<<endl; // donne itérateur sur le premier élement a
  cette clé
2    cout<<upper_bound('c')<<endl; // donne itérateur sur le dernier élement a
  cette clé
3    cout<<equal_range('c')<<endl; // donne les deux itérateurs
```

multimap II

```
1   multimap<char,double> ma;
2    ma.insert(make_pair('b',1.5));
3    ma.insert(make_pair('c',2.5));
4    ma.insert(make_pair('b',3.5));
5    ma.insert(make_pair('c',4.5));
6    for (auto el : ma){cout << el.first <<" " << el.second<<endl;}
7    cout<<ma.count('b')<<endl;
8    ma.erase('b');
9    for (auto el : ma){cout << el.first <<" " << el.second<<endl;}
```

```
user $ b 1.5
user $ b 3.5
user $ c 2.5
user $ c 4.5
user $ 2
user $ c 2.5
user $ c 4.5
```

Set

- Le conteneur ”Set” est un cas particulier du conteneur ”Map”.
- Il ne contient que des clés, pas de données associées.
- Dans un conteneur set, on ne peut plus modifier un élement inséré.

```
1 #include <set>
2 set<char> l;
3 l.insert('c');
4 l.count('c');
5 l.erase('c');
```

- le Conteneur set est ordonnés.
- Exemple d’application: une base mathématique particulière.
- Il existe multiset.