Les conteneurs associatifs

Contrairement aux séquences, les conteneurs associatifs sont capables d'identifier leurs éléments à l'aide de la valeur de leur clef. Grâce à ces clefs, les conteneurs associatifs sont capables d'effectuer des recherches d'éléments de manière extrêmement performante. En effet, les opérations de recherche se font généralement avec un coût logarithmique seulement, ce qui reste généralement raisonnable même lorsque le nombre d'éléments stockés devient grand. Les conteneurs associatifs sont donc particulièrement adaptés lorsqu'on a besoin de réaliser un grand nombre d'opération de recherche.

La table associative (map)

Une map permet d'associer une clé à une donnée.
La map prend au moins deux paramètres :

- le type de la clé
- le type de la donnée

Complexité

- Insertion : O(log(n))
- Recherche : O(log(n))

La table associative est en quelque sorte une généralisation du tableau : le tableau associe des entiers consécutifs à des valeurs d'un certain type alors que la map associe des valeurs d'un type arbitraire à des valeurs d'un autre type.
Les algorithmes de manipulation des conteneurs associatifs imposent qu'il existe une relation d'ordre pour le type de donnée utilisé pour les clefs des objets.
Comme pour les autres conteneurs que nous avons vu précédemment, les éléments stockés dans les conteneurs associatifs sont de type ***value_type***.

La class map contient donc des paires d'objet : *clé* + *données*.
Elle se déclare de la manière suivante :
map<Key, Data, Compare>.
Deux objets d'une map ont forcément des clés différentes.

Exemple :

```
#include <iostream>
#include <map>
#include <string.h>

using namespace std;

int main (){
  map<string,unsigned int> nbJoursMois;
  nbJoursMois["janvier"] = 31;
  nbJoursMois["fevrier"] = 28;
  nbJoursMois["mars"] = 31;
  nbJoursMois["avril"] = 30;
```

```
 nbJoursMois["mai"] = 31;
 nbJoursMois["juin"] = 30;
 nbJoursMois["juillet"] = 31;
 nbJoursMois["aout"] = 31;
 nbJoursMois["septembre"] = 30;
 nbJoursMois["octobre"] = 31;
 nbJoursMois["novembre"] = 30;
 nbJoursMois["decembre"] = 31;


 cout << "La table contient " << nbJoursMois.size() << " elements " <<endl;
 for (auto it=nbJoursMois.begin(); it!=nbJoursMois.end(); ++it) {
   cout << it->first << " -> \t" << it->second << endl;
 }
 cout << "Nombre de jours du mois de janvier : " << nbJoursMois["janvier"] << endl;
 map<string,unsigned int>::iterator courant  = nbJoursMois.find("mars");
 //auto courant  = nbJoursMois.find("mars");
 auto precedent = courant;
 auto suivant   = courant;
 ++suivant;
 --precedent;
 cout << "Le mois précédent (dans l'orde alphabétique) est " << (*precedent).first << endl;
 cout << "Le suivant : " << (*suivant).first << endl;

 return 0;
}
```

Le fait d'accéder à une clé via l'opérateur [ ] insère cette clé dans la map (avec le constructeur par défaut pour la donnée) si cette denière est absente de la map. Ainsi lpour vérier si une clé est présente, il faut utiliser la méthode find.

Voici quelques méthodes de la classe Map :

- bool empty ( ) const; : Teste si le conteneur ne contient pas d'éléments, retourne le résultat du test begin() == end().
- size_type size() const; : Retourne le nombre d'éléments, i.e. std::distance(begin(), end())
- void clear(); : Efface tous les éléments du conteneur. Après l'appel à cette méthode, size() retourne zéro. Invalide l'ensemble les références, pointeurs, ou itérators pointant sur des éléments du conteneur. Les itérateurs *past-the-end* ne sont pas invalidés.
- std::pair<iterator,bool> insert(const value_type& value); : Insère l'élément dans la map, si celle-ci ne contient pas déjà un élément avec une clé identique. Retourne une pair constituée d'un itérateur sur l'élément inséré (ou sur l'élément qui a empêché l'insertion) et d'un booléen précisant si l'insertion a eu lieu.

- iterator find(const Key& key); : Recherche un élément avec la clé *key*. Retourne un itérateur sur un élément de clé *key*. Si aucun élément correspondant n'est trouvé, un itérateur après le dernier élément est renvoyé .

Les ensembles d'éléments (set)

La bibliothèque standard définit également des conteneurs associatifs qui considèrent que les objets sont leur propre clef. Ces conteneurs sont appelés des ensembles.
La classe *set* est un conteneur qui stocke des éléments uniques suivants un ordre spécifique (c'est-à-dire un ensemble ordonné et sans doublons d'éléments).
La complexité est O(log(n)) pour la recherche et l'insertion.

Exemple d'utilisation :

```
#include <iostream>
#include <set>

using namespace std;

int main () {
  int tab1[ ] = {5, 2, 6, 4, 13, 10}; // non ordonné
  int tab2[ ] = {45, 21, 67, 21, 23, 45}; // non ordonné avec des doublons
  set<int> ensemble1 (tab1, tab1+4); // the range (first,last)
  set<int> ensemble2 (tab2, tab2+6); // the range (first,last)
  cout << "L'ensemble 1 contient :";
  for (set<int>::iterator it=ensemble1.begin(); it!=ensemble1.end(); ++it) {
    cout <<"  " << *it;
  }
  cout << endl;
  cout << "L'ensemble 2 contient :";
  for (set<int>::iterator it=ensemble2.begin(); it!=ensemble2.end(); ++it) {
    cout <<"  " << *it;
  }
  cout << endl;
  return 0;
}
```

Les set sont généralement utilisés dans les arbres binaires de recherche.