Analyse et Programmation Orientées Objets / C++

La classe **map** (STL)

Présentation de la classe

- **Ensemble non ordonné d'associations**
- **Initialisation par insertion d'associations**
- **Accès par la clé de chaque association**
- **Relation d'ordre obligatoire sur la clé**
- **Performances en $O(c0*\log_2(N))$ en recherche**

Créer et initialiser un dictionnaire (1)

```
#include <map>
using namespace std;
#pragma warning (disable : 4786)
#define abonne map<char*, char*>::value_type
void main () {
map <char*, char*>  annuaire;

    annuaire.insert(abonne("Durand", "04.93.77.18.00"));
    annuaire.insert(abonne("Dupuy",  "04.93.66.38.76"));
    annuaire.insert(abonne( "Leroy",  "04.92.94.20.00"));
    ---
```

Créer et initialiser un dictionnaire (2)

```
#include <map>
using namespace std;
#pragma warning (disable : 4786)
void main () {
map <char*, char*>  annuaire;

    annuaire["Durand "]= "04.93.77.18.00";
    annuaire["Dupuy"]  =  "04.93.66.38.76";
    annuaire["Leroy"]   =  "04.92.94.20.00";
    ---
```

Visualiser un dictionnaire (1)

```
---
void visualiser (char* titre, map<char*, char*>& cible) {
int taille= cible.size();
    cout << titre << ”{";
    if (taille==0) {cout << ”}"; return;}
    map<char*, char*>::iterator k= cible.begin();
    while (taille >1) {
        cout <<"<" << k->first << " : " << k->second << ">, ";
        k++;
        taille--;
    }
```

--- Suite transparent suivant

Visualiser un dictionnaire (2)

```
---
   cout <<"<" << k->first << " : " << k->second << ">";
   cout << "}";
}
```

Parcourir un dictionnaire

```
---
float moyenne (map<char*, float>& notes) {
map <char*, float>::iterator k= notes.begin();
float somme=0.0f;

   while (k != notes.end()) {
      somme += k->second;
      k++;
   }
   return somme/notes.size();
}
```

Dupliquer un dictionnaire

```
---
void main () {
map <char*, char*>  annuaire, travail;

      annuaire["Durand "]= "04.93.77.18.00";
      annuaire["Dupuy"]  =  "04.93.66.38.76";
      annuaire["Leroy"]    =  "04.92.94.20.00";
      ---
      travail= annuaire;
```

Accéder à un élément cible

```
---
void main () {
map <char*, char*>  annuaire, travail;
char* dupuy="Dupuy";

    annuaire["Durand "]= "04.93.77.18.00";
    annuaire[dupuy]      =  "04.93.66.38.76";
    annuaire["Leroy"]    =  "04.92.94.20.00";
    ---
    cout << "Numero de Dupuy :  " << annuaire[dupuy] ;
```

Comparer deux dictionnaires entre eux

```
---
void main () {
map <char*, char*>  op1, op2;
      op1["Durand "]= "04.93.77.18.00";
      op1["Dupuy"]  =  "04.93.66.38.76";
      ---
      op2["Leroy"]    =  "04.92.94.20.00";
      ---
      cout  <<  ”Resultat : " <<  (op1==op2) << endl;
```

Supprimer un élément cible

```
---
void main () {
map <char*, char*>  annuaire;
char* dupuy="Dupuy";

      annuaire["Durand "]= "04.93.77.18.00";
      annuaire[dupuy]     =  "04.93.66.38.76";
      annuaire["Leroy"]    =  "04.92.94.20.00";
      ---
      annuaire.erase(dupuy) ;
```

Supprimer les doublons consécutifs

Pas de méthode unique pour la classe map

Trier un dictionnaire

Pas de méthode sort pour la classe map

Inverser un dictionnaire

Pas de méthode reverse pour la classe map

Transférer le contenu d'un dictionnaire

```
---
void main () {
map <char*, float>  anglais, transfert;

      anglais["Durand "]=  17.5f;
      anglais["Dupuy"]  =  18;
      anglais["Leroy"]    =  4.25f;
      ---
      transfert.swap(anglais);
```

Récapitulatif partiel

- **Méthodes des conteneurs associatifs**

  begin, end, size, swap, empty, erase, ...

  Opérateurs : **=, ==, !=, []** (interne et externe)

- **Méthodes propres aux dictionnaires**

  Constructeur de copie

  find