Implémentation de table de hachage (2 séances)

Préambule

Compte tenu du temps imparti pour ce travail, choisissez d'abord des solutions faciles à mettre en œuvre, vous pourrez ensuite, une fois la structure opérationnelle, envisager des optimisations.

A. -Listes

Écrivez d'abord une structure de données générique (paramétrée par les types K et V) de listes d'associations respectant (le plus possible, commentez) la présentation ci-dessous et implémentée par un chaînage de couples, avec doublons et non trié. Renseignez bien les PRE- et POST-conditions, mentionnez l'ordre de grandeurs des coûts (bien définir la taille des données). Écrivez en parallèle au fur et à mesure un programme de test permettant de vérifier pour chaque méthode écrite son fonctionnement (tests unitaires).

```
      AListe ()                 // constructeur, crée une AListe vide
      ~AListe ()                // destructeur, libère la mémoire

void  associer(K clf,V valr)    // ajoute le couple (clf,valr)
                                //    ou change la valeur associée à clf s'il y en avait une
bool  estALVide()               // VRAI ssi aucun couple n'est stocké
V     valeurAssociée(K clf)     // donne la valeur associée à la clef clf
void  dissocier(K clf)          // supprime le couple (clf,.) ; ne fait rien s'il n'y en a pas
bool  estClef(K clf)            // teste l'existence d'un couple (clf,.)
void  trousseau(K* clfs, int ref N )   // mets les clefs présentes dans le tableau pointé par clfs
                                       // (à déclarer à l'extérieur) et mets dans N leur nombre
```

Vous trouverez sur Madoc un exemple de déclaration et d'utilisation d'une classe paramétrée en C++ (les points du plan) dans le dossier (et l'archive) pointGenerique. Attention, la classe Point étant générique, il n'est pas possible de la compiler séparément (avec l'option -c), son code est inclus par transitivité, donc `gpp testPoint.cpp -o testPoint.exe` suffit.

Fichier **point.hpp**

```
…
template < typename Nombre = float >
class Point {
 …
} ;
#include "point.tpp"
```

Fichier **point.tpp**

```
…
// chaque routine générique est préfixée par le template
template < typename Nombre >
void Point< Nombre >::deplacer(Nombre dx, Nombre dy)
{ … }
…
```

Fichier **testPoint.cpp**

```
…
#include "point.hpp" // pour utiliser le type Point

// instanciation des types génériques utilisés dans ce programme (provoque la compilation de ces classes)
template class Point<>;
template class Point<int>;

// fonction principale
int main() {
   Point<> p1; // point à coordonnées float (par défaut)
   Point<int> p2; // point à coordonnées int
 …
}
```

H. -Table

Écrivez ensuite une structure de données générique (paramétrée par les types K et V) de table de hachage respectant aussi la spécification ci-dessus (le plus possible : discutez) et implémentée par tableau de taille fixe de listes d'associations (l'implémentation précédente) . Renseignez bien les PRE- et POST-conditions, mentionnez l'ordre de grandeurs des coûts (bien définir la taille des données). Écrivez en parallèle et au fur et à mesure un programme de test permettant de vérifier pour chaque méthode écrite son fonctionnement (tests unitaires).

Vous supposerez disposer pour le type K passé en paramètre d'une fonction `int hash(K clef)` donnant un entier de hashage ; vous écrirez celles pour les types int, float, char, string afin de tester votre implémentation.