Une classe paramétrée `Dictionnaire`

Travaux dirigés et pratiques - Programmation par objets 2 (UE ULIN606)

Un dictionnaire est un ensemble d'associations, c'est-à-dire de couples (clé, valeur), où la clé est d'un type donné $TypeCle$ et la valeur d'un type donné $TypeValeur$.
Une association peut être représentée par la classe `Assoc` vue en cours.

1 Modélisation

Un dictionnaire possède habituellement les méthodes suivantes :
- `put` : place une clé et une valeur associée dans le dictionnaire ;
- `get` : prend une clé et renvoie la valeur associée ;
- `estVide` : dit si le dictionnaire est vide ;
- `taille` : retourne le nombre d'associations clé-valeur effectivement présentes dans le dictionnaire ;
- `contient` : prend une clé et dit si elle est présente dans le dictionnaire ;
- `affiche` : affiche le contenu du dictionnaire sur un flot de sortie.

2 Spécification

Écrivez la partie "signatures des méthodes" du fichier `.h` correspondant au dictionnaire générique. Ajoutez la surcharge de l'opérateur `<<` pour l'affichage et la surcharge de l'opérateur `=` pour l'affectation.

3 Implémentation par un tableau dynamique d'associations

Le tableau d'associations est alloué dynamiquement, par défaut avec une taille de 10, et augmente de 5 en 5 en cas de débordement.
L'indice d'une association est calculé grâce à une fonction de hachage appliquée à la clé (voir en annexe). Quand il faut agrandir le tableau, on redistribue les associations existantes dans le nouveau tableau en utilisant la fonction de hachage (voir en annexe aussi).
En utilisant les exemples de résultats de la fonction de hachage, faites quelques essais d'insertions "à la main" pour voir comment les choses se passent :
par exemple `put("abricot",235) ; put("amande",1023) ; put("ananas",242) ; put ("pomme",83) ; ...`

Complétez la partie "attributs" du fichier `.h`, et écrivez le fichier `.cc` correspondant.

Pour faciliter l'écriture de certaines méthodes, on peut écrire la méthode privée :

```
void CherchCl(const TypeCle& cl, int& i, int& res);
   /* cherche la cle cl dans le dictionnaire :
    si cl est presente: renvoie res=1, i indice de la case de cl dans T;
    si cl est absente et le dictionnaire non plein: renvoie res=0, i indice
      de case possible pour cl dans T;
    si cl est absente et le dictionnaire plein: renvoie res=2, i non
     significatif. */
```

4 Utilisation

Instanciez vos classes pour avoir un dictionnaire dont les clés sont des chaînes et les valeurs des entiers.

Question 1

Écrivez un programme simple qui teste le fonctionnement de ce dictionnaire (ajoutez des couples, affichez le dictionnaire, rajoutez des couples pour tester l'agrandissement, affichez, etc...).

Question 2

Utiliser la classe dictionnaire pour stocker les mots d'un texte lu sur l'entrée standard et comptabiliser le nombre d'occurrences de chacun.

.../...

5 Annexe

Fonction de hachage

Voici un exemple de fonction de hachage simple :

```
#include<string>

int hash(string s, int tailleTab)
{int i=0;
 //calcul d'un entier associe a la string
 for (int j=0; j<s.length(); j++) i=i+(j+1)*s[j];
 //adaptation a la taille du tableau
 return (i % tailleTab);
}
```

Voici quelques résultats produits par cette fonction :

taille tableau = 10	taille tableau = 15
hash(abricot, 10) = 8	hash(abricot, 15) = 13
hash(amande, 10) = 2	hash(amande, 15) = 7
hash(pomme, 10) = 2	hash(pomme, 15) = 12
hash(ananas, 10) = 3	hash(ananas, 15) = 3
hash(prune, 10) = 6	hash(prune, 15) = 1
hash(griotte, 10) = 3	hash(griotte, 15) = 13
hash(poire, 10) = 0	hash(poire, 15) = 5
hash(orange, 10) = 1	hash(orange, 15) = 1
hash(citron, 10) = 8	hash(citron, 15) = 3
hash(mangue, 10) = 6	hash(mangue, 15) = 1
	hash(papaye, 15) = 6

Lorsque la fonction *hash* produit une même valeur pour deux clés différentes, on est dans une situation de collision. La méthode de résolution de collisions la plus simple à implémenter est la méthode "recherche séquentielle simple" : on recherche, à partir de la case qui aurait du être la bonne, la première case libre (si on arrive en fin de tableau, on recommence au début), et on place l'association dedans. Cette méthode n'est pas excellente (elle crée des "grappes"), mais elle suffit pour tester cet exercice.

Agrandissement du tableau

Une bonne gestion d'un tel tableau consiste à l'agrandir dès qu'il est plein à 75%, de façon à éviter au maximum les collisions.

Dans le cadre de cet exercice, faites le contraire : laissez-le se remplir complètement, afin précisément de pouvoir observer des collisions.

Un essai que vous pouvez faire ensuite si vous avez le temps : agrandir le tableau d'un nombre aléatoire de cases, au lieu de prendre toujours 5 de plus. Ceci a l'avantage de désagréger mieux les grappes produites à l'étape précédente.