Table de hachage

Dans ce TP on implémente un table de hachage (ou table associative, ou dictionnaire) à l'aide d'un tableau de listes simplement chaînées.

Le but de l'exercice est d'implémenter une structure de données appelée *tableau associatif* (similaire aux dictionnaires de Python par exemple) afin de de compter le nombre d'apparitions de chaque mot d'un texte. On supposera qu'un mot est une suite d'au plus de 50 caractères, (lue par exemple par `scanf("%50s",mot)`).

Pour permettre un accès rapide aux mots, on utilise une méthode appelée *hachage ouvert* (ou *adressage séparé*). Elle consiste à ranger les mots dans un tableau de `N` listes chaînées, l'indice de la liste à laquelle peut appartenir un certain mot étant calculé à l'aide d'une fonction `unsigned int Hachage(char Mot[])` qui renvoie une valeur comprise entre `0` et `N - 1` (c'est à dire un indice dans le tableau).

Pour plus d'efficacité, chaque liste sera maintenue triée dans l'ordre croissant des mots. L'ordre entre 2 mots est fourni par la fonction `strcmp(char *,char *)` (consulter `man strcmp`).

La recherche d'un mot dans la table de hachage est plus efficace que dans une simple liste chaînée, car elle se ramène au calcul de la fonction de hachage puis à la recherche dans une liste (courte), plutôt qu'au parcours d'une très longue liste chaînée. Si `N` est bien choisi, on obtient alors de très bonnes performances en pratique.

On utilisera pour implémenter les listes chaînées le type suivant :

```
typedef struct celmot{
    char *mot;      /* adresse de la chaine de caracteres */
    int nombre;     /* nombre d'apparitions du mot */
    struct celmot *suivant;
} CelluleM, *ListeM;
```

1. Écrire une fonction `ListeM Allouecellule(char Mot[])`. La nouvelle cellule est créée en utilisant la place mémoire minimale pour recopier `Mot` et un nombre d'apparitions égal à 1.
2. Écrire une fonction `int AjouteListe(ListeM *L, char Mot[])` qui ajoute le mot `Mot` à la liste triée `(*L)` : si `Mot` était déjà présent dans la liste on incrémente son nombre d'apparitions, sinon on insère une nouvelle cellule en respectant l'ordre. La fonction renverra `0` en cas d'erreur, `1` en cas d'exécution correcte.
3. Écrire une fonction `unsigned int Hachage(char Mot[])` qui renvoie la somme :

$$\left(\sum_{i=0}^{n-1}(i+1)\,\texttt{Mot[i]}\right) \mod N$$

   où n est la taille de la chaîne `Mot` (obtenue par exemple par la fonction `strlen(char *)`).
4. Écrire une fonction `int RechercheDico(ListeM T[], char Mot[])` qui renvoie le nombre d'apparitions du mot `Mot` dans le tableau associatif `T`.

5. Écrire une fonction `int AjouteDico(ListeM T[], char Mot[])` qui ajoute le mot `Mot` au tableau associatif `T` s'il n'était pas présent, et incrémente son nombre d'apparitions sinon. La fonction renverra `0` en cas d'erreur, `1` en cas d'exécution correcte.

6. Écrire une fonction `int NombreMot(ListeM T[])` qui renvoie le nombre total de mots mémorisés dans `T` (en comptant toutes les apparitions de chaque mot).

7. On veut écrire un programme qui lit les mots d'un fichier texte et crée un nouveau fichier texte contenant la liste de ces mots et le nombre d'apparitions de chacun d'entre eux. Le fichier de sortie contiendra un mot par ligne, les mots apparaissant en ordre quelconque.

   La référence du fichier dans lequel on lit le texte et celle du fichier dans lequel on écrit les mots du texte seront fournis comme paramètres sur la ligne de commande. Ainsi si l'exécutable a pour nom `Compte`, l'exécution de la ligne

   ```
   $ Compte Examen Examen.cpt
   ```

   entraînera la création du fichier `Examen.cpt`. Si le contenu du fichier `Examen` est :

   ```
   Examen d'informatique tres tres facile.
   ```

   celui du fichier `Examen.cpt` devra être (à l'ordre des lignes près) :

   ```
   tres 2
   Examen 1
   d'informatique 1
   facile. 1
   ```

   Écrire la fonction `main` permettant ce traitement.

8. Comment choisir la fonction de hachage pour n'avoir qu'une seule liste chaînée ? Comment se comporte le programme sur un texte très long dans ce cas ?

9. Tester différentes fonctions de hachage, mesurer le temps d'exécution obtenu, et comparer la taille des listes construites :

   - nombre de listes vides,
   - nombre d'éléments dans la plus longue liste,
   - nombre moyen d'éléments dans les listes non vides, etc.

10. Pour les personnes très énervées : modifier le programme afin que la taille des tables de hachage ne soit plus fixée par une constante `N` mais varie dynamiquement au cours de l'exécution, en fonction de la charge de la table (nombre total de mots différents qu'elle contient). On utilisera le principe des tableaux dynamiques vu au S3.