**Université de Reims Champagne-Ardenne**
**Département de Mathématiques et Informatique**

**Pierre Delisle**
**Info0501**
**2018/2019**

# Travaux Pratiques 3
# Tables de hachage
# Analyse de texte

L'objectif de ce TP est d'implémenter un programme qui compte le nombre de mots différents dans un texte à l'aide des tables de hachage dont les collisions sont gérées par chaînage. Pour ce faire, nous allons tout d'abord implémenter une liste doublement chaînée permettant de stocker des mots.

**Partie 1 – Implémentation par liste chaînée**

1) Créez les fichiers **cellule.c** et **cellule.h** implémentant une structure de cellule permettant de stocker un mot et de la relier à un prédécesseur et à un successeur de même type. Définissez les procédures de cellule suivantes :

- **initialiserCellule** : initialise une cellule à l'aide d'un mot ;
- **détruireCellule** : libére (si nécessaire) les ressources mémoire d'une cellule.

2) Créez les fichiers **liste.c** et **liste.h** et implémentez-y une structure de liste doublement chaînée de cellules en implémentant les procédures de liste suivantes :

- **initialiserListe** : initialise une liste vide ;
- **détruireListe** : libère (si nécessaire) les ressources mémoire d'une liste ;
- **insérer** : prend en paramètre un pointeur sur une cellule et insère cette cellule au début d'une liste ;
- **rechercher** : recherche un mot dans une liste et retourne soit un pointeur sur la cellule qui contient le mot, soit NULL ;
- **supprimer** : prend en paramètre un pointeur sur une cellule d'une liste et supprime cette cellule de la liste.

Notez que pour une implémentation propre qui respecte les principes de la structure de liste, les fichiers liste.c et liste.h ne devraient pas contenir d'autres fonctions que celles demandées ci-haut. Par conséquent, tout le reste du code que vous développerez dans ce TP doit être à l'extérieur de ces fichiers et vous ne devez pas utiliser vos listes autrement que par ces fonctions (pas d'accès direct à la structure de liste à partir de l'extérieur).

3) Créez les fichiers **outilsListe.c** et **outilsListe.h** qui contiendront diverses fonctions utilitaires de manipulation des listes, et implémentez-y une fonction **afficherListe** qui affiche à l'écran les informations relatives à votre liste.

4) Créez un programme de test permettant de vérifier le fonctionnement de votre implémentation en demandant à l'utilisateur de saisir des mots, en les insérant successivement dans une liste et en affichant la liste ainsi créée. Essayez aussi de supprimer un mot, de rechercher un mot et de détruire la liste.

L'objectif du programme est de compter le nombre de mots différents dans un texte alors nous allons tenter l'exercice en utilisant une liste chaînée uniquement.

5) Récupérez le fichier **texte1.txt** sur le site web du cours (http://cosy.univ-reims.fr/~pdelisle/enseignement.php) et insérez chaque mot différent qui y figure dans votre liste chaînée. Autrement dit, pour chaque mot que vous lisez dans le fichier, recherchez-le dans la liste et insérez-le s'il n'y figure pas déjà. Pour l'instant, vous pouvez simplifier le problème en supposant qu'un mot est une chaîne de caractères précédée et suivie d'un blanc (espace ou fin de ligne). Autrement dit, vous pouvez supposer que "Quebec" et "Quebec." sont deux mots différents. Une fois tout le fichier traité, comptez le nombre d'éléments présents dans la liste en définissant la fonction **compterListe** et vérifiez la conformité du résultat.

6) Testez votre programme avec des textes de plus grande taille présents sur le site web du cours. Il est à noter que pour éviter les problèmes de conversion de caractères en C, les caractères accentués ont été remplacés (é → e, à → a, etc.) et certains caractères spéciaux (°, £, etc.) ont été éliminés. De plus, pour éviter des problèmes de débordement plus loin, ne considérez que les mots de 26 caractères et moins. Donnez le nombre total de mots présents dans le fichier, le nombre de mots différents et le temps d'exécution requis pour compter le nombre de mots différents.

Comme vous avez sûrement pu le remarquer, les fichiers les plus longs à traiter sont les dictionnaires. Afin de traiter ces derniers plus rapidement, nous allons tenter d'améliorer notre analyseur de texte à l'aide des tables de hachage.

**Partie 2 – Implémentation par table de hachage**

Nous allons expérimenter l'utilisation d'une table de hachage avec gestion des collisions par chaînage. Nous aurons donc besoin d'un tableau de listes chaînées.

7) Créez les fichiers **tableHachage.h** et **tableHachage.c** et implémentez-y une structure adéquate pour une table de hachage ainsi que les fonctions suivantes :

- **initialiserTableHachage :** initialise une table de hachage de dimension spécifique ;
- **detruireTableHachage** : libère (si nécessaire) les ressources mémoire d'une table de hachage.

8) Créez **outilsTableHachage.h** et **outilsTableHachage.c** et implémentez-y la fonction **afficherTableHachage**.

Nous devons maintenant implémenter l'insertion de mots dans notre table de hachage. Pour ce faire, nous devons être en mesure de convertir une chaîne de caractères en un entier $k$ en s'assurant que des chaînes différentes soient associées à des entiers différents. Pour ce faire, on peut interpréter une chaîne comme un entier exprimé dans une base adaptée :

1. on interprète la chaîne de caractères comme un ensemble d'entiers correspondant au code ASCII de chaque caractère (A = 65, B = 66, Z = 90, a = 97, b = 98, 0 = 48, 1 = 49, …) en respectant leur ordre.
   Par exemple, "Abc" → {65, 98, 99} ;
2. partant du principe que le codage ASCII de base utilise 128 caractères, on exprime la chaîne en base 128 en additionnant les valeurs de tous les entiers exprimés dans cette base. Ainsi, le dernier entier est multiplié par $128^0$, l'avant-dernier par $128^1$, et ainsi de suite, puis toutes les valeurs obtenues sont finalement additionnées.
   Par exemple, "Abc" → $65 * 128^2 + 98 * 128^1 + 99 * 128^0 = 1\,064\,960 + 12\,544 + 99 = 1\,077\,603$.

9) Écrivez une fonction **convertirChEntier** qui prend une chaîne de caractères et retourne l'entier $k$ associé tel que défini ci-haut (attention aux débordements).

Maintenant que nous sommes en mesure de convertir une chaîne de caractères en entier, nous pouvons l'insérer dans notre table de hachage en utilisant une fonction de hachage appropriée.

10) Écrivez une fonction **hachage** qui détermine un indice $h(k)$ à partir d'un $k$ donné par la méthode de la division.

11) Complétez l'implémentation de votre table de hachage en définissant les fonctions suivantes :

- **insérerHachage** : prend en paramètre un pointeur sur une cellule contenant une chaîne de caractères et l'insère à l'endroit approprié dans une table de hachage ;
- **rechercherHachage** : recherche une chaîne de caractères dans une table de hachage et retourne soit un pointeur vers la cellule qui contient la chaîne, soit NULL ;
- **supprimerHachage** : prend en paramètre un pointeur sur une cellule d'une table de hachage et supprime cette cellule de la table.

12) Testez votre implémentation sur l'ensemble des fichiers de texte : insérez tous les mots dans une table de hachage où $m = 11$ et comptez ensuite le nombre de mots présents dans la table en définissant la fonction **compterTableHachage**. Vérifiez la similarité des résultats avec votre implémentation en liste et comparez leur temps d'exécution.

**<u>Questions supplémentaires pour les rapides et les motivés</u>**

- Implémentez votre analyseur de texte en utilisant des listes triées en ordre alphabétique et comparez la performance avec votre implémentation initiale ;
- Écrivez un programme qui compte le nombre d'occurrences de chaque mot d'un texte ;
- Améliorez votre programme pour gérer les ponctuations (points, virgules, apostrophes, etc.).