# Programmation C
# TP 12

Table de hachage.

L'objectif de ce TP est de compter le nombre de mots différents dans un texte relativement long. Nous allons commencer par une méthode naïve puis ensuite nous utiliserons une fonction de hachage pour augmenter la vitesse de traitement.

Télécharger le fichier `Germinal.txt` sur le moodle de l'université. C'est dans ce fichier que l'on souhaite compter le nombre de mots différents. Pour modéliser nos ensembles de mots, nous utiliserons des listes chaînées. Ainsi, la structure C suivante sera centrale dans ce TP :

```
1 typedef struct node{
2   char* word;
3   struct node* next;
4 }Node, *List;
```

Une cellule est composée d'un mot et d'un pointeur vers une cellule suivante. Une liste est ici un pointeur vers une première cellule. Un ensemble de mot sera une liste chaînée dans laquelle les mots contenus dans les cellules sont tous différents.

Pour ne pas réinventer la roue, il sera crucial d'utiliser les fonctions de la librairie `string.h`.

**Exercice 1** **Comptage naïf**

Commencer par compter les mots du fichier texte `Germinal.txt` en lisant le fichier mot par mot puis en les ajoutant à une seule liste chaînée lorsque ces derniers ne sont pas dedans.

Pour mener à bien cette mission, les tâches atomiques à implémenter sont les suivantes :

- Allocation d'une cellule avec un mot spécifié en argument.
- Recherche d'un mot dans une liste chaînée de mots.
- Insertion d'une cellule en tête de liste chaînée.
- Calcul de la longueur d'une liste chaînée.
- Parcage d'un fichier ouvert mot par mot (n'allez pas chercher plus loin qu'un `fscanf` dans un `while`...)

Une fois votre exécutable fonctionnel, tester rapidement la durée d'exécution avec la fonction `time` du terminal :

```
nborie@perceval$ time ./simplelist
25353 different words found in Germinal
```

```
real 0m10.873s
user 0m10.855s
sys 0m0.008s
nborie@perceval$
```

## Exercice 2 Comptage par paquets

Nous allons maintenant tenter une approche plus performante. Une première chose à comprendre est que l'approche naïve coûte chère car elle nécessite un très grand nombre d'appels à la fonction `strcmp` (du moins, si vous n'avez pas réinventé la roue...).

Pour réduire ce nombre d'appels, nous allons utiliser une fonction de hachage. C'est une fonction qui associe un nombre à chaque mot (les langages objets associe un nombre à chaque objet pour faciliter les manipulations). Prenons la fonction suivante, qui à un mot `char* w` associe le nombre suivant:

$$hash(w) := \left( \sum_{i=0}^{i=strlen(w)} (i+1) * w[i] \right) \% \, NB\_PACK$$

où `NB_PACK` a été défini sous forme de macro en tête de fichier comme il suit `#define NB_PACK 4096`. Par exemple, pour le mot "mot" qui est composé de trois lettres, sachant que dans le code ascii, 'm' vaut 109, 'o' vaut 111 et 't' vaut 116, on aura:

$$hash(''mot'') = ((0+1)*109 + (1+1)*111 + (2+1)*116)\%4096 = 679\%4096 = 679$$

Pour faire un comptage utilisant cette fonction de hachage, définissez dans votre programme un tableau de `NB_PACK` listes chaînées vides pour commencer. Ensuite, lorsque l'on lira le fichier mot par mot, on commencera par calculer la valeur de hachage du mot et on inséra le mot si l'on ne l'a déjà vu dans la liste chaînée indexée dans le tableau de liste par la valeur de hachage. Ainsi le mot "mot" devra être inséré dans la liste chaînée indexée par 679.

Ce tableau de listes chaînées est ce qu'on appelle une table de hachage. Le gain est situé dans le fait qu'au lieu de comparer tous les mots, on commence par comparer les valeurs de hachage. Une bonne fonction de hachage est une fonction qui se calcule rapidement et qui prend de manière uniforme toutes les valeurs possibles. Au final, votre programme devrait faire des `strcmp` que lorsque plusieurs mots différents ont la même valeur de hachage et on a justement choisie cette fonction de manière à ce que le phénomène arrive rarement.

Faites une mesure rapide du temps de calcul et comparez.

```
nborie@perceval$ time ./hachage
25353 different words found in Germinal

real 0m0.036s
user 0m0.036s
sys 0m0.000s
nborie@perceval$
```