TP 4 : tables de hachage

On se propose dans ce TP de stocker un ensemble de mots dans une table de hachage avec gestion des collisions par chaînage externe. Le fichier `motsAscii` se trouvant à l'adresse

`www.lif.univ-mrs.fr/~fdenis/algoL2/motsAscii`

contient 1349 mots du français, dont les accents ont été enlevés pour des raisons de difficultés d'encodage. Chaque lettre peut être représentée par un entier compris entre 0 et 255, son code Ascii. Et chaque mot peut être représenté comme un mot en base 256. Par exemple, le mot `blanc` est représenté par le nombre

$$98 \cdot 256^4 + 108 \cdot 256^3 + 97 \cdot 256^2 + 110 \cdot 256 + 99.$$

Les mots du fichier ont tous une longueur inférieure ou égale à 14. Les mots du français ont tous une longueur inférieure ou égale à 25. Par précaution, on peut considérer une borne supérieure égale à 30. Désignons par `LONGTABLE`, la longueur de la table de hachage, par `BASE` l'entier 256. Cela conduit aux définitions suivantes :

```
#define LONGMAX 30 // longueur maximale d'un mot
#define LONGTABLE 240 // longueur de la table de hachage
#define BASE 256 // un octet

typedef char MOT[LONGMAX];

typedef struct maillon *LISTE_MOT;

typedef struct maillon{
      MOT val;
      LISTE_MOT suiv;
}MAILLON;


LISTE_MOT Table[LONGTABLE]; // table de hachage : variable globale
```

1. Écrivez une fonction `int hash1(MOT mot)` qui calcule le reste de la division de `mot` par `LONGTABLE`. Bien entendu, il est hors de question de calculer la valeur numérique correspondant à `mot` : elle peut être de l'ordre de $256^{14} = 2^{112} = 4(2^{10})^{11} \simeq 4 \cdot 10^{33}$ qui dépasse très largement l'arithmétique exacte proposée par un langage comme le C. En revanche, on peut utiliser les égalités suivantes pour écrire cette fonction en restant dans le domaine des `int` : soit $w_n = a_0 b^n + a_1 b^{n-1} + \ldots + a_n$. On a $w_n = b w_{n-1} + a_n, w_0$ et donc
$$w_n \bmod m = [b(w_{n-1} \bmod m) + a_n] \bmod m.$$

2. Écrivez une fonction `void InitTable(void)` qui initialise toutes les listes de la variable globale `Table` à `NULL`.
3. Écrivez une fonction `void insererTable(MOT mot)` qui insère le mot passé en argument en tête de la liste `Table[hash1(mot)]`.
4. Écrivez une fonction `void insererFichier(char *adr)` qui insère tous les mots du fichier situé à l'adresse passée en argument dans la table `Table`. Votre programme pourra ressembler à

```
void insererFichier(char *adr){
  FILE *f;
  char s[LONGMAX];
  f=fopen(adr,"r");
  while (fscanf(f,"%s",s)!=EOF){
    insererTable(s);
  }
  fclose(f);
}
```

5. Si la liste `Table[c]` contient n_c enregistrements, le temps moyen de recherche d'un enregistrement dans cette liste est de $(n_c + 1)/2$. Le temps moyen de recherche d'un enregistrement dans la table est donc égal à

$$\frac{1}{N}\sum_c n_c(n_c + 1)/2$$

où N est le nombre total d'enregistrement : $N = \sum_c n_c$. Écrivez une fonction `float TempsMoyRech()` qui calcule ce temps moyen une fois que la variable `Table` est remplie.

6. Calculez ce temps moyen pour différentes valeurs de `LONGTABLE : 256, 240, 253`. Si les N enregistrements étaient également répartis entre les différentes positions du tableau, on aurait $n_c = N/LONGTABLE$ et un temps moyen de recherche qui serait égal à

$$\frac{N}{2 \cdot LONGTABLE} + \frac{1}{2}.$$

Expliquez cette formule, calculez les temps moyens optimaux pour `LONGTABLE = 256, 240, 253` et commentez les résultats.

7. La méthode de hachage par multiplication repose sur la formule suivante :

$$hash2(c) = \lfloor LONGTABLE \cdot \{\alpha c\} \rfloor$$

où $\alpha \in]0, 1[$ et où $\{x\} = x - \lfloor x \rfloor$ désigne la partie fractionnaire de x. En vous inspirant de la question 1, et en remarquant que $\{\alpha(a + b \cdot c)\} = \{\alpha a + b \cdot \{\alpha c\})\}$, écrivez la fonction `int hash2(MOT mot)`.

8. Testez les performances de cette méthode pour différentes valeurs de $\alpha : 0.5, 0.1, 0.01, (\sqrt{5}-1)/2$ (pour cette dernière valeur, comparez les formats `float` et `double`).