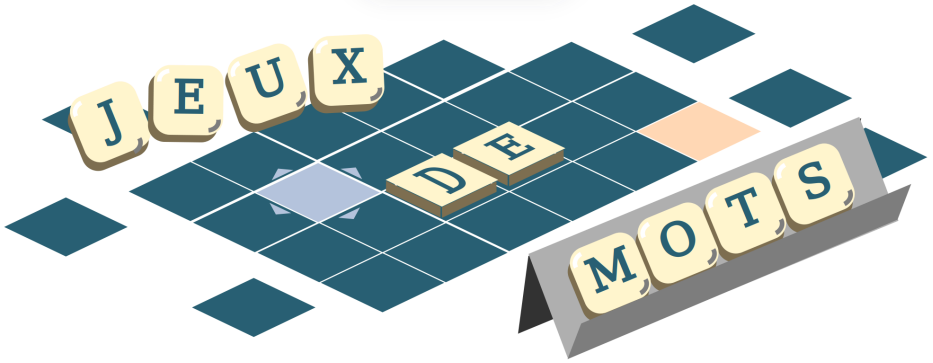
# TP n°11







[1]

## I. Quelques propriétés sur les mots

1. Implémenter une fonction de prototype `bool sont_miroirs(char* mot1, char* mot2)` qui vérifie si deux mots sont de même longueur et sont miroirs l'un de l'autre. Par exemple `"SAPER"` et `"REPAS"` sont miroirs l'un de l'autre.

2. En déduire une implémentation d'une fonction de prototype `bool est_palindrome(char* mot)` qui vérifie si un mot donné est un palindrome, comme `"ICI"`, `"RADAR"` ou `"ENGAGELEJEUQUEJELEGAGNE"`.

3. Implémenter une fonction de prototype `int indice_lettre(char c)` qui renvoie l'indice d'une lettre majuscule (avec la convention `'A'` -> 0, `'B'` -> 1, `'C'` -> 2, ..., `'Z'` -> 25). On pourra utiliser un cast de la forme `int v = (int) c;` pour interpréter le caractère `c` comme un entier. Tester.

4. Implémenter une fonction de prototype `bool sont_anagrammes(char* mot1, char* mot2)` qui vérifie si deux mots sont anagrammes, c'est-à-dire contiennent exactement les mêmes lettres avec les mêmes multiplicités. On utilisera deux tableaux statiques de 26 cases pour compter le nombre d'occurrences de chaque lettre dans les deux mots, avant de comparer ces deux tableaux.

[1] Je dédie ce TP à **Baptiste Mouillon**, qui m'a raconté cette petite blague à l'époque (parmi d'autres). Bravo à toi pour ton titre de champion du monde de Scrabble francophone, et ton record du plus haut ELO de tous les temps !



## II. Récupération d'un lexique

### Un lexique sous forme de liste chaînée

L'objectif de cette partie est de construire un **lexique**, c'est-à-dire un ensemble de mots, à partir d'un fichier. Un tel lexique sera implémenté par une liste simplement chaînée dans laquelle chaque maillon contient un pointeur vers une chaîne de caractères (représentant un mot) allouée dans le tas.

5. Faire un schéma mémoire d'une telle structure de lexique, par exemple avec trois mots.

6. Proposer et implémenter un type structuré adapté pour cette liste, dont chaque maillon possède en particulier un champ de type `char*`. Renommer ce type en `lexique`.

7. Implémenter des fonctions élémentaires permettant de manipuler ces listes chaînées : allouer un nouveau maillon en tête de liste, libérer convenablement la structure et afficher le lexique en entier. Attention notamment à la libération, on demande ici de libérer la liste chaînée, mais également de libérer les chaînes de caractères dans les maillons (avant de libérer le maillon bien sûr).

### Le fichier "Officiel du Scrabble"

L'**Officiel du Scrabble** est un dictionnaire de référence utilisé pour arbitrer les compétitions officielles de Scrabble. C'est un fichier pertinent pour pouvoir récupérer des mots de la langue française. Aller sur cahier-de-prepa.fr/mp2i-liv et télécharger le fichier nommé **ods6.txt** dans **TPs/Ressources**. Il s'agit d'un fichier contenant un mot par ligne, écrit en lettres majuscules et sans aucun accent ou caractère exotique. Placer ce fichier dans le même dossier que votre code C pour la suite.

**On pourra supposer qu'aucun mot de ce fichier ne fait 30 caractères ou plus.**

On cherche à lire ce fichier afin de construire un lexique. Pour cela on utilisera un tableau statique de 30 caractères comme tampon pour temporairement stocker la chaîne lue par `fscanf`.

8. Comment peut-on utiliser la fonction `fscanf` pour récupérer la chaîne de caractères contenue dans une ligne du fichier ? Comment peut-on savoir que l'on est arrivé à la fin du fichier et qu'il n'y a plus rien à lire ?

9. Implémenter une fonction de prototype `lexique* construit_lexique(char* nom_de_fichier)` qui lit un fichier donné ligne par ligne avec la fonction `fscanf`. On donnera à `fscanf` l'adresse d'un tableau statique de 30 caractères pour y recopier le mot lu sur la ligne. Puis, avant de passer au mot suivant, on calculera la taille du mot avec une fonction bien choisie de la bibliothèque `<string.h>` (cf. TP6 si besoin). On allouera alors un tableau de caractères de bonne taille sur le tas pour y recopier le mot, puis on ajoutera un maillon en tête de liste, avec un pointeur vers cette chaîne de caractères fraîchement allouée sur le tas. Afficher votre lexique pour vérifier que tout est bien récupéré.

## III. Curiosités de la langue française

### Consigne

Pour chacune des questions suivantes, il est demandé d'écrire une ou plusieurs fonctions pour trouver la réponse.

**10** Parcourir le lexique français pour trouver quelle est la dernière anagramme du mot `"ISOMETRIES"`, et comprendre la blague.

**11** Quel est le plus long palindrome de la langue française ?

**12** Un **isogramme** est un mot dans lequel chaque lettre apparaît au plus une fois, comme par exemple `"COMPILATEUR"`. Donner un plus long isogramme de la langue française.

## IV. Un solveur pour le jeu du Scrabble

**13** Implémenter une fonction de prototype `int score(char* mot)` qui renvoie le score d'un mot au Scrabble. On utilisera un tableau statique contenant le nombre de points de chaque lettre, trouvable sur Internet (par exemple la lettre `'A'` vaut 1 point et la lettre `'B'` vaut 3 points).

**14** Implémenter une fonction de prototype `bool peut_etre_ecrit(char* mot, char* lettres)` qui vérifie si un mot donné peut-être écrit à l'aide des lettres présentes dans le mot `lettres`. Attention : la multiplicité des lettres compte ! On pourra utiliser des tableaux qui comptent les occurrences.

**15** Implémenter une fonction de prototype `void affiche_meilleur_coup(lexique* l, char* lettres)` qui parcourt le lexique pour trouver le mot français qui rapporte le plus de points et qui peut être construit avec les lettres données, puis qui l'affiche avec son score.

**16** Dans le jeu du Scrabble, il existe des tuiles **joker**, qui peuvent remplacer n'importe quelle lettre. Proposer une variante de la fonction précédente dans laquelle on prend également un nombre de jokers en argument. Les jokers ne rapportent aucun point mais permettront d'obtenir plus de mots.



## V. Vers l'idée d'une table de hachage

On dit qu'un mot est **réversible** si son mot miroir pris en lisant les lettres de droite à gauche est aussi un vrai mot de la langue française. Tous les palindromes sont notamment réversibles, mais l'autre implication n'est pas vraie. Dans cette dernière partie, on cherche un mot réversible le plus long de la langue française.

**17** Implémenter une fonction de prototype `bool est_reversible(lexique* l, char* mot)` qui prend un lexique et un mot, et vérifie si le miroir de ce mot appartient au lexique ou pas.

**18** Implémenter une fonction `plus_grand_reversible` qui parcourt le lexique, et pour chaque mot rencontré teste s'il est réversible en appelant la fonction précédente, pour finalement renvoyer le plus grand mot réversible trouvé.

**19** Lancer votre programme en appelant cette fonction pour chercher le plus long mot réversible de la langue française. Que se passe-t-il ? Justifier cela grâce à la complexité temporelle pire cas de votre fonction et du nombre de mots. Vous pouvez utiliser Ctrl + C pour arrêter un processus.

### Partitionner les mots via leur taille

Lorsque l'on veut tester si un mot comme `"AME"` est réversible, on parcourt tout le lexique en entier. Pourtant, il n'y a aucune chance qu'un mot de taille 4 comme `"YEUX"` soit le miroir du mot. De manière générale les seuls candidats envisageables sont ceux de taille exactement 3. Il est donc inutile de chercher dans le lexique en entier.

Ainsi, on propose désormais de partitionner les mots en $k$ lexiques $l_1, ..., l_k$ différents. Chaque lexique $l_i$ contient tous les mots de taille exactement $i$, et uniquement ceux-là. Les $k$ pointeurs vers ces lexiques seront stockés dans un tableau de $k$ cases alloué sur le tas. On appelle alors la fonction `plus_grand_reversible` sur chaque lexique $l_i$.

**20** Implémenter une fonction `construit_lexiques`, très similaire à la fonction `construit_lexique`, mais qui cette fois-ci alloue sur le tas un tableau de 30 pointeurs vers des lexiques, initialise chaque case avec le pointeur `NULL`, puis ajoute les mots lus issus du fichier dans le bon lexique $l_i$ à chaque fois. Par exemple, si dans le fichier on lit le mot `"STEAK"`, on l'ajoute au lexique $l_5$ car ce mot possède 5 lettres. Cette fonction doit finalement renvoyer ce tableau contenant des pointeurs vers les 30 lexiques ainsi construits et dont l'union correspond à tous les mots du fichier.

**21** Implémenter une fonction permettant de connaître la taille d'une liste simplement chaînée.

**22** Implémenter une fonction qui appelle `construit_lexiques` pour récupérer un tableau de 30 lexiques, puis affiche la taille de chaque lexique $l_i$. Combien de mots sont de taille exactement 7 ? Et de taille exactement 10 ?

**23** Utiliser cette approche pour déterminer un mot réversible le plus long en français, en ne cherchant que des candidats au sein du même lexique $l_i$. Peut-on observer un gain de temps ?