Perfectionnement à la programmation C
Compte-rendu TP 6

Chayma Guerrassi & Irwin Madet

24 mars 2020

1 Schéma fonctionnel du projet

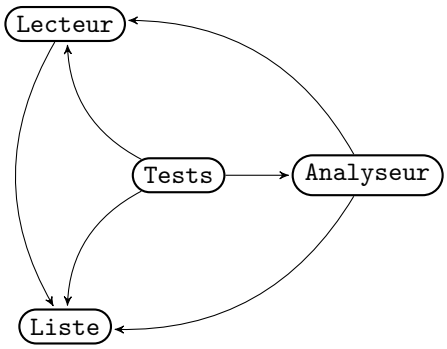

FIGURE 1 – Graphe des dépendances

1.1 Module `Liste`

Le module `Liste` se charge de stocker les mots en correspondance avec leur nombre d'occurrences. Il se charge également de déclarer les types `Cell` et `List` (sachant que le type `List` est un pointeur de type `Cell*`).

Ce module inclut les fonctions suivantes :

- `int insertWord(List *list, char *word)` : Cette fonction permet d'insérer un mot en tête de liste si le mot n'y est pas présent. Dans le cas où le mot est déjà dans la liste, son nombre d'occurrences est incrémenté.
- `int insertWordAlphabetically(List *list, char *word)` : Cette fonction insert le mot dans une liste par ordre lexicographique.
- `int insertWordByOccurrences(List *list, char *word)` : Cette fonction insert le mot dans une liste par ordre croissant d'occurrences.
- `void freeList(List *list)` : Cette fonction se charge de libérer la mémoire utilisée par une liste. Elle est appelée à la fin du programme.
- `void displayList(List list)` : Cette fonction affiche la représentation d'une liste sur la sortie standard. Elle est très utile lors du développement pour déboguer le programme.
- `Cell *getWord(List list, char *word)` : Cette fonction permet de cherche un mot dans une liste. Si le mot n'est pas dans la liste, la fonction retourne `NULL` sinon elle retourne la cellule correspondante.

1.2 Module `Lecteur`

Le module `Lecteur` se charge de lire un fichier, et d'en extraire les mots. Il se charge également de déclarer le type énumération `SortType`.

Ce module inclut les fonctions suivantes :

- `int readFileByWords(FILE *file, List *list, SortType sort)` : Cette fonction lit un fichier mot par mot et les insert dans une liste selon un type de tri.

1.3 Module `Analyseur`

Le module `Analyseur` se charge de regrouper les mots ensemble par préfixes ou suffixes d'un mot donné ou par expressions de longueur n.

- `int groupBySuffixes(List wordsList, List *list, char *word, SortType sort)` : Cette fonction récupère la liste de tous les mots du texte et les groupe par suffixes du mot `word`. Le tout est trié selon le mode de tri `sort`.
- `int groupByPrefixes(List wordsList, List *list, char *word, SortType sort)` : Cette fonction récupère la liste de tous les mots du texte et les groupe par préfixes du mot `word`. Le tout est trié selon le mode de tri `sort`.
- `int groupByExpressions(List wordsList, List *list, int n, SortType sort)` : Cette fonction récupère la liste de tous les mots du texte et les groupe par expressions de `n` mots. Le tout est trié selon le mode de tri `sort`.

1.4 Module `Tests`

Le module `Tests` se charge de tester les différentes fonctions du projet afin de s'assurer que chaque fonction a bien le comportement qu'on attend d'elle. Pour cela, les fonctions sont appelées avec des données définies et dont on connait à l'avance le résultat que l'on devrait obtenir de la fonction.
Il possède une unique fonction `int test()` dans son entête qui exécute tous ses tests et retourne 1 si tout s'est bien déroulé ou 0 si un test a échoué. Chaque test qui échoue envoie une erreur sur la sortie standard pour savoir duquel il s'agit.

2 Modification ultérieures

Il avait été décidé dans un premier temps de faire une table de hachage pour stocker les mots plus facilement. Cependant, ce système s'est avéré trop complexe et pas assez flexible. De plus il ne répondait pas du tout à l'exercice demandé. Nous avons donc opté pour des insertions triées, et dont la position est mise à jour à chaque insertion du même mot.

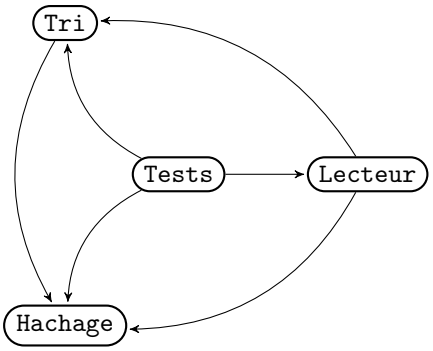

FIGURE 2 – Graphe des dépendances imaginé au départ.

En ce qui concerne les tests, nous avons rapidement chercher un moyen de faire une boucle pour appeler toutes les fonctions de test au lieu de les appeler en dur dans le code. C'est ainsi que nous avons découvert que les fonction de type `int` sont en réalité de type `int *`. Il faut donc faire un tableau de type `int (*tab[])()`.

Lors du développement de la fonction `insertWordByOccurrences()`, nous avons cherché un moyen simple mais efficace de mettre à jour un mot dans la liste. C'est alors que nous avons voulu décaler les cellules dans la liste en direction de la tête. Pour ce faire, nous avons revu la structure pour y ajouter un nouvel élément : un pointeur vers la case précédente. Nous utilisons dès lors des listes doublement chaînées. Ceci permet beaucoup de flexibilité et permet même de lire une liste dans les deux sens. Il a cependant fallu revoir toutes les autres fonctions du module pour correctement gérer cette nouvelle implémentation.