Rapport de TP AI01
ABOU GHALIA Mostafa et MARRAS Hugo

On note n le nombre d'éléments qui compose la chaîne de caractère str
et m le nombre d'éléments dans la liste de type List

Fonctions demandées:

1- *initialize :*
- Prend en entrée un pointeur vers un élément de type List
- Aucune sortie car void
- Complexité : **Θ**(1)

2- *insert_empty_list* :
- Prend en entrée un pointeur vers un élément de type List et un pointeur vers une chaîne de caractère
- Aucune sortie car void
- Complexité : **O**(2n) => **Θ**(n)

3- *insert_begining_list :*
- Prend en entrée un pointeur vers un élément de type List et un pointeur vers une chaîne de caractère
- Aucune sortie car void
- Complexité : **O**(2n) => **Θ**(n)

4- *insert_end_list :*
- Prend en entrée un pointeur vers un élément de type List et un pointeur vers une chaîne de caractère
- Aucune sortie car void
- Complexité : **Θ**(2n) => **Θ**(n)

5- *insert_after_position* :
- Prend en entrée un pointeur vers un élément de type List,un pointeur vers une chaîne de caractère et un entier
- Sortie 0 ou -1 en fonction de la réussite de l'ajout (0 -> réussite , -1 -> échec)
- Complexité : **Ω**(n) et **O**(n*m)

6- *remove_elem:*
- Prend en entrée un pointeur vers un élément de type List et un entier
- Sortie 0 ou -1 en fonction de la réussite de la suppression (0 -> réussite , -1 -> échec)
- Complexité : **O**(m*n) et **Ω(n)**

7- *compare :*
- Prend en entrée deux pointeurs vers chaines de caractères (str1 et str2)
- Sortie 1 ou 2 en fonction de l'élément le plus grand (1 -> str1, 2 -> str2)
- Complexité : **Ω**(n) et **O**(m)

8- *sort* :
- Prend en entrée un pointeur vers un élément de type List
- Sortie 0 ou -1 (-1 si liste vide)
- Complexité : **Ω**(n*m) et **O**(n^2*m)

9- *display* :
- Prend en entrée un pointeur vers un élément de type List
- Aucune sortie car void
- Complexité : **Θ**(n*m)

10- *destruct* :
- Prend en entrée un pointeur vers un élément de type List
- Aucune sortie car void
- Complexité : **Θ**(n*m)

Fonctions supplémentaires:

freeElement :
- Prend en entrée un pointeur vers un élément de type 'Element'
- Aucune sortie car void
- Complexité : **Θ**(n)

ReconstructionStr :
- Prend en entrée un pointeur vers un élément de type 'Element' et pointeur vers une chaîne de caractère
- Aucune sortie car void
- Complexité : **Θ**(n)

Separation :
- Prend en entrée un pointeur vers une chaîne de caractères
- Sortie : pointeur vers un élément de type 'Element'
- Complexité : **Θ**(n)

nombreValide :
- Prend en entrée un pointeur vers une chaîne de caractères
- Sortie : Entier 1 ou 0 (1 si nombre valide, 0 sinon)
- Complexité : **Θ**(n)

retirerZeros :
- Prend en entrée un pointeur vers une chaîne de caractères
- Sortie : Pointeur vers une chaîne de caractère
- Complexité : **Ω**(1) et **O**(n)

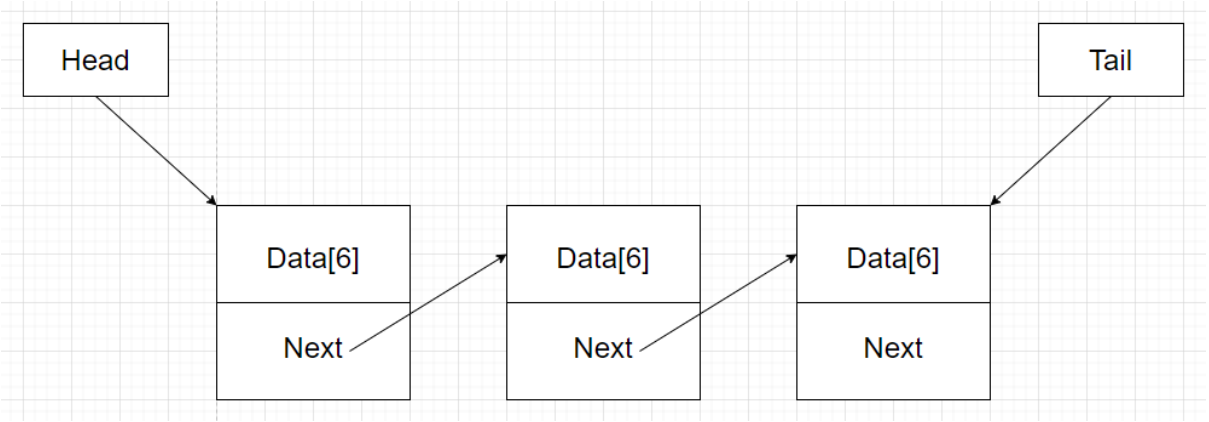
dernierElement:
- Prend en entrée un pointeur vers un élément de type Element
- Sortie : Pointeur vers un élément de type Element
- Complexité : **Θ**(n)

sum :
- Prend en entrée un pointeur vers une liste type List
- Sortie : aucune
- Complexité : **Θ**(m*n)

Rapport de TP AI01
ABOU GHALIA Mostafa et MARRAS Hugo

Explication du fonctionnement du code schématisé :

Voici le schéma de la structure général du code que nous avons utilisé pour modéliser les données :



Réflexion sur le projet :

On pourrait créer une variable NombreElement et NombreStr dans la structure de la liste indiquant le nombre d'Élément et le nombre de nombre stockés dans la liste et ainsi que la position de tous les éléments, cela permettrait de simplifier certaines fonctions.

On pourrait transformer la structure en liste doublement chaînée, ce qui permettrait d'éviter d'utiliser de la mémoire dans certaines fonctions pour stocker des valeurs temporaires.

On pourrait créer une structure DataElement qui permettrait de stocker tous les Str séparé par 5 dans un seul élément. Nous avons effectué un schéma et produit une version V2 du code avec ce fonctionnement.