TD/TP Structure de données - Listes chaînées

M. Tellene

1 Appréhender la structure de données liste chaînée

Exercice 1

Décrire l'état de la liste chaînée après chaque ligne du programme suivant :

```
lc = ListeChainee()
lc.insertion_en_tete(1)
lc.insertion_en_queue(8)
lc.insertion_en_tete(2)
lc.insertion_element(3,1)
lc.suppression_en_tete()
lc.suppression_en_queue()
lc.modifier_ieme_element(12,1)
```

Exercice 2

Écrire un programme qui crée une liste chaînée. Le programme devra faire 3 ajouts en tête des valeurs 8 puis 1 puis 9. Une fois fait, le programme devra faire 2 ajouts en queue des valeurs 5 puis 1.

Sur votre feuille, décrire l'état de la liste chaînée.

Reprenez le programme et modifiez l'élément à l'indice 3 par la valeur 4. Faire une suppression en tête et une suppression en queue.

Sur votre feuille, re-décrire l'état final de la liste chaînée.

2 Manipuler une liste chaînée

Organisation : aller dans votre dossier personnel, puis créer un dossier « structure de données » (ou « SD »), dans ce dossier, créer un dossier « liste chainee », c'est ce dossier qui contiendra le travail fait lors de ce TP.

On rappelle qu'une liste chaînée est composée de maillon (ou de cellule). Un maillon est définie pour une `info` et par un maillon `suivant`.

Vous commencerez par créer une classe `Maillon`, il est à noter que la classe `Maillon` ne possède qu'un constructeur.

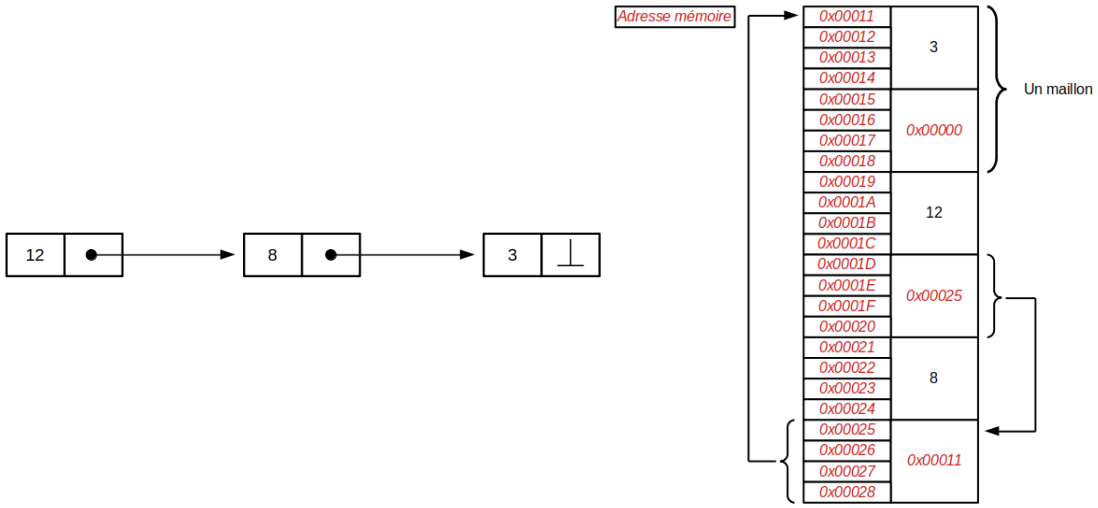
Sous la classe `Maillon`, vous créerez une classe `ListeChaine`. Une liste chaînée est définie par une `tete` et une `queue`, ces deux éléments sont initialisés à `None` lors de la création de la liste chaînée.

Une fois fait, créer les méthodes associées à la structure de données de liste chaînées. Si vous ne vous en souvenez pas, elles sont données dans la suite.

— `afficher()` : les valeurs des maillons sont affichées
— `est_vide()` : renvoie `True` si la liste est vide, `False` sinon
— `insertion_en_tete(x)` : ajoute un maillon ayant la valeur `x` en tête de liste (penser au cas où l'on ajoute en tête dans une liste chaînée vide)
— `insertion_en_queue(x)` : ajoute un maillon ayant la valeur `x` en queue de liste (penser au cas où l'on ajoute en queue dans une liste chaînée vide)
— `vider()` : la liste ne contient plus aucun maillon
— `rechercher_element(x)` : renvoie l'indice du maillon ayant pour valeur `e`, la méthode renverra -1 si l'élément n'est pas trouvé ou si la liste chaînée est vide
— `ieme_element(i)` : renvoie la valeur du maillon à l'indice `i`
— `modifier_ieme_element(x, i)` : la valeur du maillon à l'indice `i` devient `x` (si `i` est plus grand que la longueur de la liste chaînée, la méthode renverra `None`)
— `suppression_en_tete()` : supprime le maillon en tête de liste
— `suppression_en_queue()` : supprime le maillon en queue de liste
— `insertion_element(x, i)` : un maillon ayant pour valeur `x` est ajouté à l'indice `i`

2.1 Comment parcourir une liste chaînée ?

Une liste chaînée est constituée de **maillon**. Un maillon est un bloc mémoire composée de deux parties : une valeur et une adresse mémoire. L'adresse correspond à l'adresse mémoire de l'élément suivant de notre maillon. Une liste chaînée se parcourt en utilisant cette adresse. Ci-dessous un exemple de liste chaînée avec la représentation en mémoire.



3 Des méthodes supplémentaires

Exercice 3
Écrire une méthode `longueur` qui renvoie la taille d'une liste chaînée. Le but de cette méthode est de parcourir la liste, du premier maillon au dernier, en suivant les liens qui relient les maillons entre eux.

Exercice 4
Écrire une méthode `nb_occ` qui renvoie le nombre d'occurrences d'une valeur `x` dans une liste chaînée.

Exercice 5
Écrire une méthode `dernier_maillon` qui renvoie le dernier maillon d'une liste chaînée.