## Entraînement : listes chaînées

Les solutions peuvent être rédigées en **pseudo-code, python, C, C++ ou Java**. La syntaxe du langage n'a pas d'importance tant que celle-ci reste **cohérente** et **compréhensible**. (Dans les exemples, les solutions sont données en pseudo-code).

**Grille d'évaluation**

| | |
|---|---|
| A (20) | L'algorithme répond au problème posé de façon claire et exhaustive. |
| B (16) | Le principe général de l'algorithme est le bon. Cependant, il y a une ou deux erreurs / oublis sur les cas particuliers ou les conditions d'arrêts et vérification de pointeurs nuls : ces erreurs impactent l'algorithme à la marge. |
| C (11) | Le principe général de l'algorithme est le bon mais les erreurs font que l'algorithme ne fonctionne pas dans de nombreux cas. |
| D (8) | Le principe général de l'algorithme ne permet pas de répondre au problème, cependant les opérations de manipulation sur la liste chaînée sont écrites correctement. Ou alors, même si le principe semble le bon, il y a de très nombreuses erreurs dans l'écriture. |
| E (1) | L'algorithme est faux ou inexistant et la structure de liste est mal utilisée. |

**Exercice 1.**

Une **file** est une structure de données "First In, First out" : on fait sortir les éléments dans l'ordre dans lequel ils sont arrivés. Elle accepte deux opérations **enfile** qui rajoute un élément et **defile** qui supprime l'élément le plus ancien.

Exemple, si l'on part d'une file vide `F` :

```
enfile(F,2)
enfile(F,1)
enfile(F,3)
enfile(F,4)
Affiche defile(F)
Affiche defile(F)
Affiche defile(F)
Affiche defile(F)
```

affiche 2134.

Et

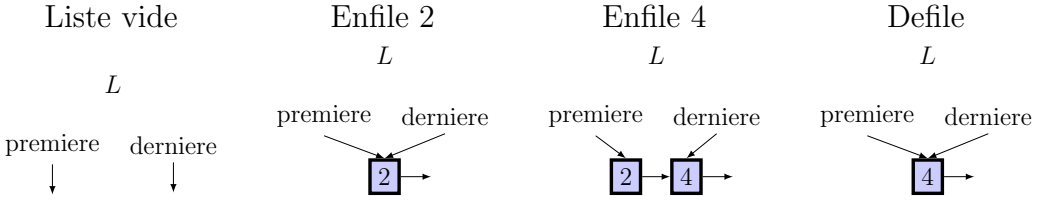
```
enfile(F,2)
enfile(F,1)
Affiche defile(F)
enfile(F,3)
enfile(F,4)
Affiche defile(F)
enfile(F,5)
Affiche defile(F)
Affiche defile(F)
Affiche defile(F)
```

affiche 21345.

On propose d'implanter une File en utilisant une structure de liste chaînée. Cependant, contrairement à d'habitude, on va maintenir **deux pointeurs** : un sur la tête de liste et un sur le dernier élément de la liste.

```
Structure Cellule :
    Entier valeur
    Cellule suivante

Structure File :
    Cellule premiere
    Cellule derniere
```

On enfile en fin de liste et on défile en début de liste.
Exemple illustré.




Implantez les fonctions `Enfile(File F, Entier a)` (pas de valeur de retour) et `Defile(File F)` (renvoie un entier). On supposera que l'appel `Cellule(a)` permet de créer une cellule de valeur $a$.

**Solution**

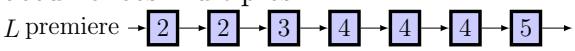
```
Enfile
Input : File F, Entier a
Procédé :
    d <- F.derniere
    n <- Cellule(a)
    Si d != None :
        d.suivante <- n
        F.derniere <- n
    Sinon :
        F.premiere <- n
        F.derniere <- n

Defile
Input : File F
Output : un entier
Procédé :
    p <- F.premiere
    Si p = None :
        Erreur
    Sinon
        v <- p.valeur
        F.premiere <- p.suivante
        Si F.premiere = None :
            F.derniere <- None
        Retourner v
```

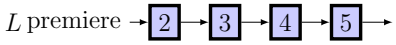
**Exercice 2** (Partiel 2017-18)**.**
On utilise la structure de donnée suivante :

```
Structure Cellule :
    Entier valeur
    Cellule suivante
```

```
Structure Liste :
    Cellule premiere
```

Donner un algorithme qui prend en entrée une liste chaînée dont on suppose les valeurs ordonnées et qui supprime les occurrences multiples.

Par exemple, si la liste est 


Le résultat après le passage de l'algorithme sera 


**Solution**

Un algorithme possible (ce n'est pas le seul)

```
Input : Une liste L
Procédé :
        c <- L.premiere
        Si c != None :
            Tant que c.suivante != None :
                v <- c.valeur
                Si c.suivante.valeur == v :
                    c.suivante <- c.suivante.suivante
                Sinon :
                    c <- c.suivante
```

Question donnée au partiel 1 2017-2018, résultats obtenus :

| A | B | C | D | E |
|---|---|---|---|---|
| 28.6% | 19% | 42.9% | 9.5% | 0% |

Exemple d'un algorithme qui obtient "B" :

```
c <- L.premiere
Tant que c.suivante != None :
   v <- c.valeur
   Si c.suivante.valeur == v :
       c.suivante <- c.suivante.suivante
   Sinon :
       c <- c.suivante
```

Oubli de tester si `L.premiere` est `None` avant de faire `c.suivante` : l'algo ne fonctionne pas sur la liste vide.

Exemple d'un algorithme qui obtient "C" (le plus courant)

```
c <- L.premiere
Si c != None :
   Tant que c.suivante != None :
       v <- c.valeur
       Si c.suivante.valeur == v :
           c.suivante <- c.suivante.suivante
       c <- c.suivante
```

On passe systématiquement à `c.suivante` : cet algorithme ne fonctionne pas dès qu'il y a plus de 2 copies d'une même cellule.