# Bases de la Programmation Impérative

**Examen de session normale 1A**
**Année 2022–2023**

**Durée : 2h00**
**Ce sujet est composé de 4 pages.**

## Consignes générales

On recommande d'essayer de traiter toutes les questions de l'énoncé : vous pouvez utiliser la fonction demandée dans une question précédente même si vous n'avez pas réussi à l'implanter.

Tous les dessins ou commentaires facilitant la compréhension de votre code seront les bienvenues.

Toutes les fonctions demandées doivent être implantées selon un modèle itératif : vous ne devez pas utiliser la récursivité dans cette épreuve.

Il n'est pas nécessaire d'utiliser des fonctions de la bibliothèque standard Python pour répondre aux questions de cette épreuve, à l'exception de la fonction `len(struct)` qui renvoie la taille de la structure de données passée en paramètre.

On impose pour chaque exercice le coût attendu de la fonction demandée : **les implantations ne respectant pas cet ordre de grandeur seront pénalisées**.

On impose dans certains exercices des contraintes supplémentaires (*i.e.* ne pas allouer de tableau, ne pas utiliser telles fonctions, *etc.* ) : **les réponses ne respectant pas ces contraintes ne rapporteront aucun point**.

# Manipulations de tableaux

Dans les deux exercices suivants on travaille sur des tableaux. Par tableau nous entendons des `list` Python sur lesquelles nous ne faisons **que des accès indexés** pour lire ou écrire la valeur d'une case. Autrement dit, on s'interdit de modifier la taille de la liste et donc d'utiliser des méthodes comme `append`, `insert`, `del`, *etc.*

## Ex. 1 : Echange de deux éléments d'un tableau *(2 pts)*

Écrire une fonction `echanger(tab, idx1, idx2)` qui échange le contenu des cases du tableau `tab` dont les indices `idx1` et `idx2` sont passés en paramètres. Cette fonction n'a aucun effet si les indices sont égaux ou si le tableau contient moins de 2 cases.

**Cette fonction doit avoir un coût constant (*i.e.* pas de parcours du tableau).**

## Ex. 2 : Problème du drapeau belge *(5 pts)*

Dans les années 1970, Edsger Dijkstra a proposé dans un de ses livres le problème du drapeau hollandais. Ce problème consiste à réorganiser les éléments d'un tableau dont les valeurs correspondent à l'une des trois couleurs d'un drapeau. L'objectif est de ranger les valeurs du tableau de la manière suivante :

- en premier toutes les valeurs correspondant à la couleur de gauche du drapeau (le noir pour nous avec le drapeau de la Belgique que nous avons choisi pour changer un peu) ;
- en second toutes les valeurs correspondant à la couleur du milieu du drapeau (le jaune pour nous) ;
- en dernier toutes les valeurs correspondant à la couleur de droite du drapeau (le rouge pour nous).

Écrire une fonction `trier_drapeau(tab)` qui réorganise les éléments du tableau `tab` afin de remplir l'objectif ci-dessus. `tab` contient uniquement des chaînes de caractères de taille 1 représentant les couleurs du drapeau belge : `"n"` pour noir, `"j"` pour jaune et `"r"` pour rouge.

La fonction renvoie deux indices (sous la forme d'un tuple) à la fin de son exécution : l'indice de la première case contenant la valeur `"j"`, et l'indice de la dernière case contenant la valeur `"j"`.

Par exemple, si le tableau initial contient `["j", "r", "n", "j", "n", "r", "r"]`, alors le tableau final **doit être** `["n", "n", "j", "j", "r", "r", "r"]` et les indices renvoyés **seront** 2 et 3.

On pose comme pré-condition que **le premier élément du tableau initial est forcément un `"j"`** (ce qui implique que le tableau initial contient au moins un élément et qu'il y a bien au moins un "j" dans le tableau).

**Cette fonction ne doit pas allouer de nouveau tableau** : elle doit réorganiser les éléments du tableau initial en faisant uniquement des échanges de valeurs grâce à la fonction `echanger` précédente.

**Vous ne devez pas utiliser des fonctions telles que `sort` ou `sorted`** : la seule fonction de la bibliothèque standard Python autorisée est `len(tab)` pour récupérer la taille du tableau.

**Cette fonction doit avoir un coût linéaire en fonction du nombre d'éléments du tableau.**

# Manipulations de listes chainées

Dans les exercices suivants, on travaille sur des listes simplement chaînées. Ces listes simplement chainées sont composées de cellules définies par la classe ci-dessous :

```
class Cellule:
    """
      Une cellule est composée d'une valeur et d'une référence vers la
       cellule suivante (ou None s'il n'y a pas de suivant)
    """
    def __init__(self, val, suiv):
        """
          Constructeur
        """
        self.val = val
        self.suiv = suiv
```

Les valeurs contenues dans les cellules sont des entiers.

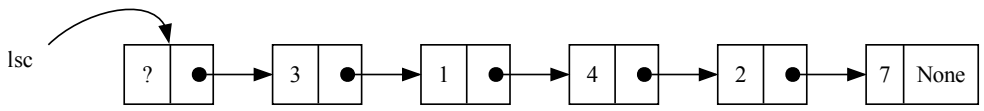
Les listes simplement chaînées manipulées ici sont équipées d'un élément fictif en tête appelé sentinelle qui ne contient aucune valeur significative et dont le champ `suiv` pointe vers la première cellule contenant une valeur significative.

**Nous n'utiliserons pas de classe `ListeSimplementChainee`**, une liste simplement chaînée sera donc simplement **représentée par la `Cellule` en tête de chaînage**.

Une liste simplement chaînée vide peut donc être créée simplement grâce à la fonction suivante :

```
def creer_liste_vide():
    """
      Renvoie une liste simplement chaînée vide.
      Elle contient en fait un élément fictif en tête (sentinelle)
        dont le champ suivant est None.
      La sentinelle a une valeur non-significative : on n'a pas le droit
        de se baser sur la valeur de cette cellule dans les fonctions
    """
    return Cellule('?', None)
```

La liste simplement chainée `lsc` contenant les valeurs 3, 1, 4, 2 et 7 peut donc être représentée par le schéma abstrait ci-dessous :




On notera que les listes simplement chaînées manipulées dans les exercices suivants peuvent être vides.

## Ex. 3 : Inversion d'une liste simplement chaînée *(3 pts)*

Écrire une fonction `inverser(lsc)` : cette fonction doit inverser la liste simplement chainée `lsc` passée en paramètre, en procédant uniquement par modification des chaînages (*i.e.* sans créer de nouvelles cellules et sans changer aucune valeur).

Par exemple, si la liste initiale est `3` → `1` → `4` → `2` → `7` → `FIN`, la liste inversée sera `7` → `2` → `4` → `1` → `3` → `FIN`.

La fonction ne renvoie rien car la tête de liste (c'est à dire l'élément fictif en tête) ne change pas : c'est son champ `suiv` qui change pour pointer vers la nouvelle première cellule contenant une valeur significative (de 3 à 7 dans l'exemple ci-dessus).

**Cette fonction doit avoir un coût linéaire en fonction du nombre d'éléments dans la liste.**

## Ex. 4 : Recherche du maximum dans liste *(5 pts)*

Écrire une fonction `rechercher_prec_max(lsc)` : cette fonction recherche et renvoie une référence vers la cellule précédant celle ayant la valeur maximale de la liste simplement chainée `lsc` passée en paramètre. Si `lsc` contient plusieurs fois la valeur maximale, la fonction renvoie la cellule qui précède la première cellule ayant la valeur maximale.

Par exemple, si la liste est `2` → `1` → `5` → `8` → `3` → `FIN` alors la fonction renvoie une référence vers la cellule contenant la valeur 5.

Si le maximum est dans la première cellule de la liste simplement chaînée `lsc`, la fonction renvoie une référence vers la cellule fictive en tête.

**Cette fonction doit avoir un coût linéaire en fonction du nombre d'éléments de la liste.**

## Ex. 5 : Tri d'une liste *(5 pts)*

Écrire une fonction `trier(lsc)` : cette fonction trie par ordre croissant la liste simplement chainée `lsc` passée en paramètre, en utilisant l'algorithme du tri par recherche du maximum. Le principe de cet algorithme est le suivant :

1. on définit une nouvelle référence qui pointera sur la première cellule du chaînage trié, qu'on appelle `res`, et qui initialement est `None` ;
2. on recherche le maximum dans `lsc` ;
3. on insère ce maximum en tête de `res` puis on le retire de là où nous l'avons trouvé dans `lsc` ;
4. on reprend à l'étape 2 tant que `lsc` n'est pas vide, sinon on passe à l'étape 5 ;
5. on finit en chaînant `res` à l'élement fictif en tête de `lsc`, qui représentera donc bien finalement la liste triée.

Vous devez bien sûr utiliser la fonction `rechercher_prec_max` de l'exercice précédent dans l'implémentation de la deuxième étape de l'algorithme.

**Cette fonction doit avoir un coût quadratique en fonction du nombre d'éléments de la liste.**