Exercice 1 (4 points)

On s'intéresse à l'évaluation d'expressions mathématiques, comportant uniquement des additions et des multiplications. On utilisera pour cela les structures de file et de pile, dont les interfaces sont données ci-dessous.

Interface de la classe `Pile`

`Pile()` : crée une pile vide
`empile(el)` : empile l'élément `el` au sommet de la pile
`depile()` : supprime et renvoie l'élément au sommet de la pile
déclenche une erreur si la pile est vide
`est_vide()` : renvoie `True` si la pile est vide,
`False` sinon

Interface de la classe `File`

`File()` : crée une file vide
`enfile(el)` : ajoute l'élément `el` à la queue de la file
`defile()` : supprime et renvoie l'élément en tête de la file
déclenche une erreur si la file est vide
`est_vide()` : renvoie `True` si la file est vide,
`False` sinon

Afin de tenir compte des priorités des opérations, on représente ces expressions par des arbres binaires.
Ainsi, l'expression $1 + 5 \times (3 + 9)$ est représentée par l'arbre suivant.

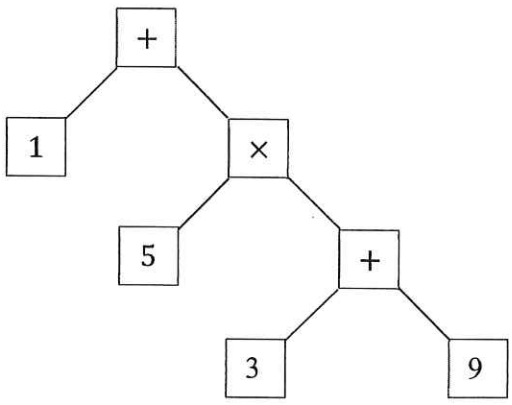

Figure 1

1. Donner l'expression représentée par l'arbre ci-dessous.

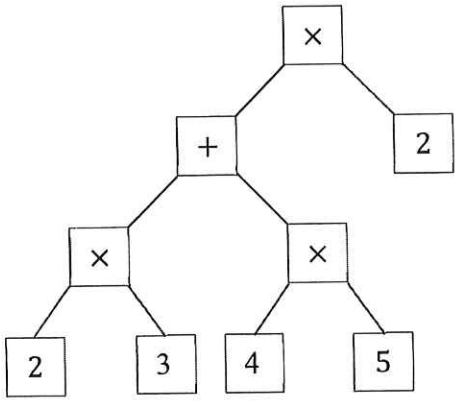

Figure 2

2. On décide d'implémenter en Python un arbre binaire à l'aide de la classe `Noeud` ci-dessous :

```
class Noeud:
    def __init__(self, etiquette, gauche, droit):
        self.etiq = etiquette
        self.sag = gauche
        self.sad = droit
```

Un sous-arbre vide sera représenté par `None`.
Dessiner l'arbre `expression` qui est défini par le code suivant.

```
feuille2 = Noeud("2", None, None)
feuille3 = Noeud("3", None, None)
feuille4 = Noeud("4", None, None)
feuille5 = Noeud("5", None, None)
feuille6 = Noeud("6", None, None)
noeud0 = Noeud("*", feuille2, feuille3)
noeud1 = Noeud("+", feuille5, feuille6)
noeud2 = Noeud("+", noeud0, feuille4)
expression = Noeud("*", noeud2, noeud1)
```

3. Le parcours suffixe (aussi appelé postfixe) d'un arbre représentant une expression mathématique permet d'en obtenir une représentation appelée notation polonaise inversée.

   a. Donner la liste des étiquettes de l'arbre de la question **1** (**Figure 2**) dans l'ordre du parcours suffixe de cet arbre.

   b. On donne ci-après trois propositions de fonctions récursives dont le premier paramètre est un arbre représentant une expression mathématique et le deuxième est une file initialement vide. Laquelle de ces fonctions renvoie la file contenant les étiquettes de l'arbre dans l'ordre du parcours suffixe ?

Proposition 1

```
def suffixe(arbre, file):
    if arbre != None:
        file.enfile(arbre.etiq)
        parcours_g = suffixe(arbre.sag, file)
        parcours_d = suffixe(arbre.sad, file)
    return file
```

Proposition 2

```
def suffixe(arbre, file):
    if arbre != None:
        parcours_g = suffixe(arbre.sag, file)
        file.enfile(arbre.etiq)
        parcours_d = suffixe(arbre.sad, file)
    return file
```

Proposition 3

```
def suffixe(arbre, file):
    if arbre != None:
        parcours_g = suffixe(arbre.sag, file)
        parcours_d = suffixe(arbre.sad, file)
        file.enfile(arbre.etiq)
    return file
```

4. L'évaluation d'une expression mathématique consiste à effectuer les différentes opérations pour obtenir le résultat du calcul correspondant.

On donne un algorithme qui permet d'évaluer une expression donnée sous la forme d'un arbre :

- On effectue un parcours suffixe de cet arbre pour obtenir une file contenant ses étiquettes dans l'ordre de la notation polonaise inversée.
- Pour chaque élément défilé,
  - si c'est un nombre, on l'empile ;
  - si c'est un opérateur (`'+'` ou `'*'`), on dépile les deux éléments `d` et `g` au sommet de la pile, et on empile le résultat de l'opération appliquée à `g` et `d`.
- Lorsque la file est vide, la pile contient un seul élément : le résultat de l'évaluation de l'expression.

Par exemple, lors de l'évaluation de l'expression en notation polonaise inversée 3 10 + 5 ×, voici les différents états de la pile, suite au défilement d'un élément :

Elément défilé	3	10	+	5	×
Pile	3	10 3	13	5 13	65

La fonction `evalue(arbre)` implémente cet algorithme. Elle renvoie le résultat de l'évaluation d'une expression mathématique représentée par un arbre binaire `arbre` passé en paramètre.

Sur votre copie, recopier et compléter cette fonction.

On pourra utiliser la fonction `op` définie ci-dessous :

```
def op(symbole, x, y):
    if symbole == '+':
        return int(x) + int(y)
    else:
        return int(x) * int(y)
```

```
def evalue(arbre):
    f = suffixe(arbre, File())
    p = Pile()
    while ... :
        elt = f.defile()
        if elt == '+' or elt == '*':

            ... # plusieurs lignes

        else:
            ...
    return ...
```