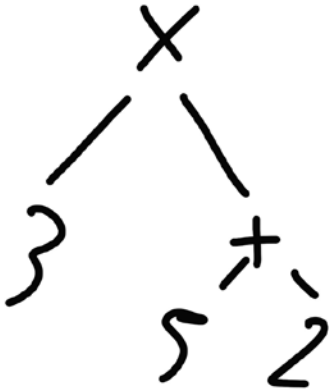
T NSI	Structures de données	*TP_POO : Parcourir les arbres*

I. Expression arithmétique



Q1.Dessiner l'arbre1

Q2. Ecrire le parcours Préfixe de l'arbre 2. Prefixe : +*3+5x*5x

Q3. Infixe : (3*(5+x))+(5*x)

Q4. Ecrire le parcours postfixe de l'arbre 2 : 3 5 x + * 5 x * +

Q5.Ajouter une méthode postfixe.

```
def postfix_traversal(self):
    s =  ""
    if self.fils_gauche is not None:
        s  +=  self.fils_gauche.postfix_traversal()
    if self.fils_droit is not None:
        s  +=  self.fils_droit.postfix_traversal()
    s  +=  str(self.valeur)
    if  self.est_feuille():
        return s
    return s
```

Q6.Ajouter une méthode prefixe.

```
def prefix_traversal(self):
    s =  ""
    s  +=  str(self.valeur)
    if self.fils_gauche is not None:
        s  +=  self.fils_gauche.prefix_traversal()
    if self.fils_droit is not None:
        s  += self.fils_droit.prefix_traversal()
    if  self.est_feuille():
        return s
    return s
```

Q7. Ecrire une méthode evaluation qui prend l'arbre d'une expression ne contenant que des constantes en argument et retourne la valeur de celle-ci.

```
def evaluer(self):
    expression = self.postfix_traversal()
    print(expression)
    return self.convertir(expression)
```

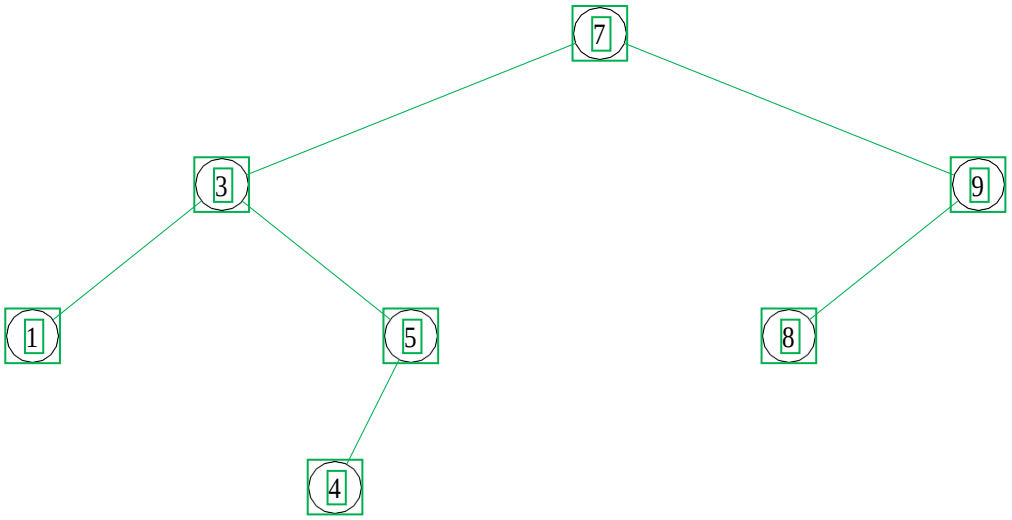
```
def convertir(self, expression):
    pile = []
    for char in expression:
        if char == '+':
            b, a = pile.pop(), pile.pop()
            pile.append(a+b)
        elif char == '*':
            b, a = pile.pop(), pile.pop()
            pile.append(a * b)
        elif char == '-':
            b, a = pile.pop(), pile.pop()
            pile.append(b-a)
        elif char == '/':
            b, a = pile.pop(), pile.pop()
            pile.append(b/a)

        else:
            pile.append(int(char))
    return pile.pop()
```

II. Le tri du bijoutier :

On dispose d'une liste de nombres. Par exemple, la liste 7, 9, 3, 5, 4, 1, 8.

On associe à chaque élément n de la liste un nœud v (initialisation : père[v]=NIL, fils_gauche[v]=NIL, fils_droit[v]=NIL, clef[v]=n).

1. Dresser l'arbre obtenu en appliquant l'algorithme Arbre_Insérer aux éléments de la liste (dans l'ordre de la liste) en partant d'un arbre vide pour le premier élément, chaque appel à l'algorithme modifiant l'arbre.
2. L'arbre obtenu :



3. L'un des parcours postfixe, infixe, préfixe de la liste trie la liste. Lequel ?

   Le parcours infixe trie la liste. Les éléments de gauche sont en effet par construction plus petits qu'un noeud et sont affichés avant le nœud dans l'ordre infixe et les éléments de droite qui sont, par construction, plus grands sont affichés dans l'ordre infixe après le nœud. Par "récurrence", on a donc un affichage des éléments de la liste dans l'ordre.