

## 8.1

## Algorithme des arbres binaires

NSI TLE - JB DUTHOIT

Notations pour les algorithmes : Soit T un arbre :

**T.racine** est le nœud racine de l'arbre T

Soit un nœud x :

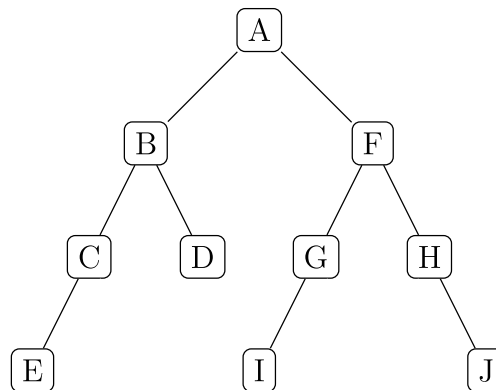
**x.gauche** correspond au sous-arbre gauche du nœud x

**x.droit** correspond au sous-arbre droit du nœud x

**x.cle** correspond à la clé du nœud x

## 8.1.1 Calcul de la taille d'un arbre

On considère de nouveau cet arbre :



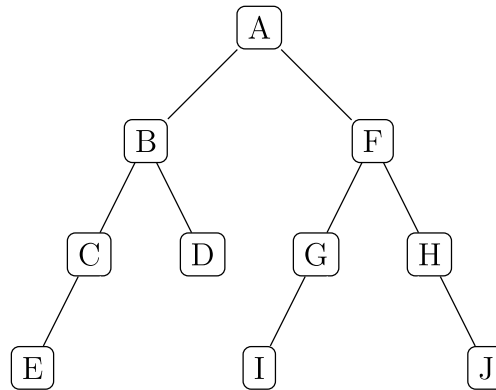
Appliquer cet algorithme à l'arbre ci-dessus.

```

1 VARIABLE
2 T : arbre
3 x : nœud
4 DEBUT ;
5 Function TAILLE(T)
6   if T ≠ NIL then
7     x ← T.racine
8     renvoyer 1 + TAILLE(x.gauche) + TAILLE(x.droit)
9   else
10    renvoyer 0
11  end
12 end
13 FIN
  
```

**Exercice 8.105**

Créer une fonction **taille(a)** qui renvoie le nombre de noeuds de l'arbre binaire a

**8.1.2 Calcul de la hauteur d'un arbre**

Appliquer cet algorithme à l'arbre ci-dessus.

```

1 VARIABLE
2 T : arbre
3 x : nœud
4 DEBUT
5 Function HAUTEUR(T)
6   if T ≠ NIL then
7     x ← T.racine
8     renvoyer 1 + max(HAUTEUR(x.gauche), HAUTEUR(x.droit))
9   else
10    renvoyer 0
11  end
12 end
13 FIN
  
```

**Exercice 8.106**

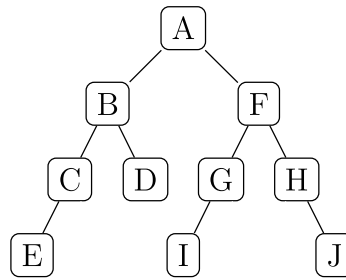
Créer une fonction **hauteur(a)** qui renvoie la hauteur de l'arbre binaire a

**8.1.3 Parcours d'un arbre binaire**

On veut ici afficher les différentes valeurs de contenues dans tous les n œuds de l'arbre, par exemple une par ligne. L'ordre dans lequel est effectuée la lecture est donc très important.

**Parcours infixe**

Un parcours **infixe** fonctionne comme suit : On parcourt le sous arbre de gauche, puis on affiche sa racine, et enfin on parcourt le sous arbre droit.



```

1 VARIABLE
2 T : arbre
3 x : noeud
4 DEBUT
5 Function PARCOURS__INF(T)
6   if T ≠ NIL then
7     x ← T.racine
8     PARCOURS__INF(x.gauche)
9     affiche x.cle
10    PARCOURS__INF(x.droit)
11  end
12 end

```

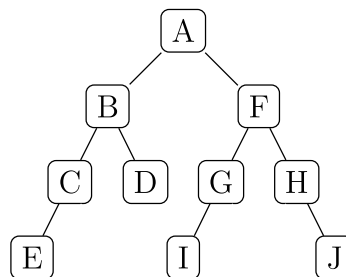
Dans quel ordre a été parcouru cet arbre ?

### ● Exercice 8.107

Créer une fonction **parcours\_\_infixe(a)** qui effectue un parcours infixe de l'arbre binaire a

## Parcours de l'arbre ordre préfixe

Un parcours **préfixe** fonctionne comme suit : On affiche la racine et on parcourt les sous arbres.



```

1 VARIABLE
2 T : arbre
3 x : noeud
4 DEBUT
5 Function PARCOURS__PREFIXE(T)
6   if T ≠ NIL then
7     x ← T.racine
8     affiche x.cle
9     PARCOURS__PREFIXE(x.gauche)
10    PARCOURS__PREFIXE(x.droit)
11  end
12 end

```

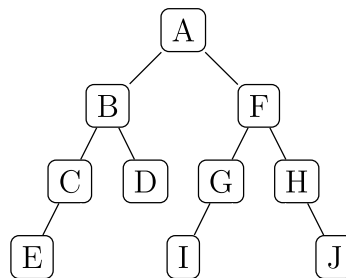
Dans quel ordre a été parcouru cet arbre ?

● **Exercice 8.108**

| Créer une fonction **parcours\_\_prefixe(a)** qui effectue un parcours prefixe de l'arbre binaire a

**Parcours d'un arbre ordre suffixe**

Un parcours **postfixe** fonctionne comme suit : on parcourt les sous arbres et on affiche sa racine.



```

1 VARIABLE
2 T : arbre
3 x : nœud
4 DEBUT
5 Function PARCOURS__SUFFIXE(T)
6   if T ≠ NIL then
7     x ← T.racine
8     PARCOURS__SUFFIXE(x.gauche)
9     PARCOURS__SUFFIXE(x.droit)
10    affiche x.cle
11  end
12 end
  
```

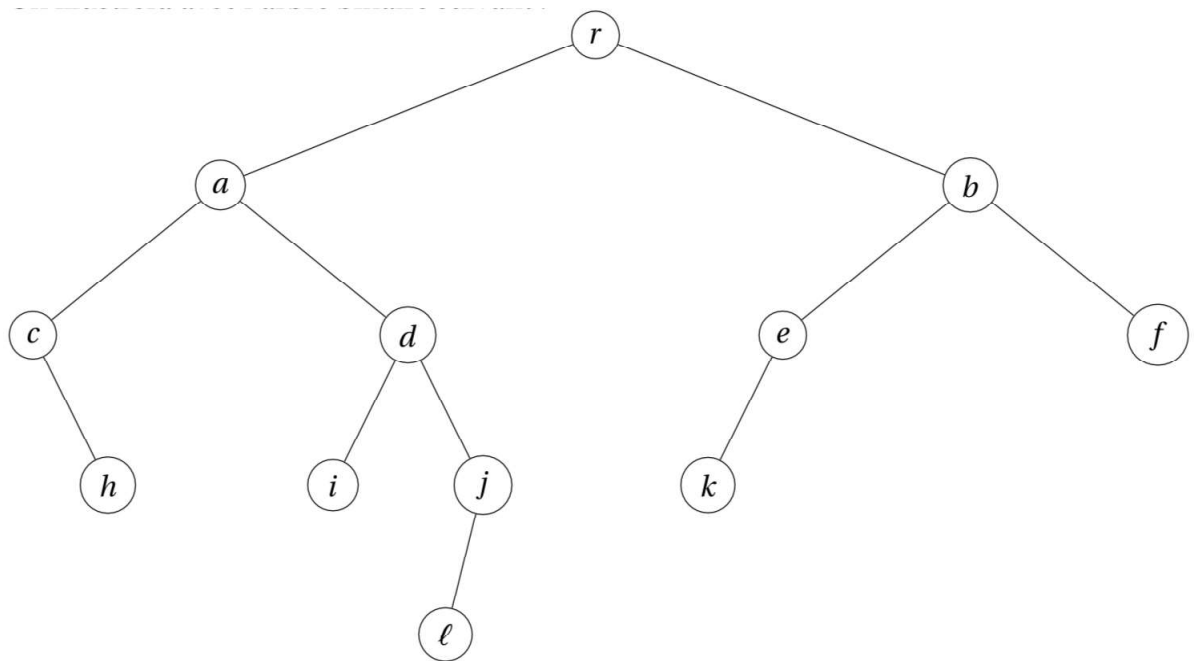
Dans quel ordre a été parcouru cet arbre ?

● **Exercice 8.109**

| Créer une fonction **parcours\_\_suffixe(a)** qui effectue un parcours suffixe de l'arbre binaire a

● **Exercice 8.110**

| On considère l'arbre binaire suivant :



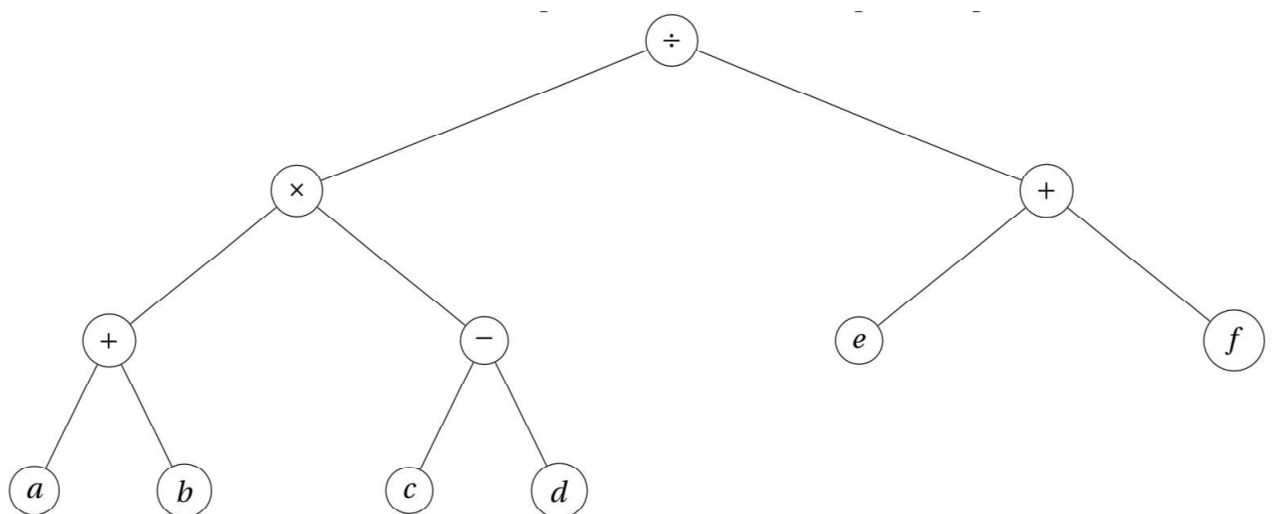
Déterminer :

1. La taille de l'arbre
2. La hauteur de l'arbre
3. Son arité
4. L'ordre infixe
5. L'ordre préfixe
6. L'ordre suffixe

\*\*\*

### Exercice 8.111

Cet exercice porte sur la notation polonaise inverse.  
On considère l'arbre binaire suivant :

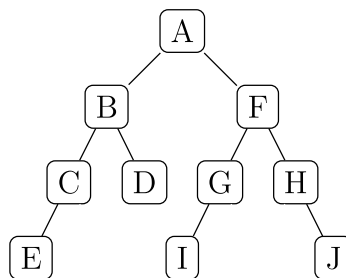


Déterminer :

1.
  - a) La taille de l'arbre
  - b) La hauteur de l'arbre
  - c) Son arité
  - d) L'ordre infixe
  - e) L'ordre préfixe
  - f) L'ordre suffixe
2. Pour le parcours infixe, on ajoute la convention suivante : On ajoute une parenthèse ouvrante à chaque fois qu'on entre dans un sous-arbre et on ajoute une parenthèse fermante dès que l'on quitte ce sous arbre. Pour les feuilles, on ne met pas de parenthèses.
  - a) Avec cette convention de parenthèses dans l'ordre infixe, donner l'expression obtenue ➤ Cette convention est indispensable pour lever toute ambiguïté. Ce n'est pas le cas pour les deux autres ordres :
  - b) L'ordre préfixe consiste à voir les opérateurs comme fonctions de deux variables, avec la fonction écrite à gauche. Écrire l'expression.
  - c) L'ordre postfixe consiste à voir les opérateurs comme fonctions de deux variables, avec la fonction écrite à droite. Écrire l'expression.

\*\*\*

### Parcourir un arbre en largeur d'abord



```

1 VARIABLE
2 T : arbre
3 Tg : arbre
4 Td : arbre x : nœud
5 f : file
6 Function PARCOURS_LARGEUR(T)
7   result ← tableau vide
8   if T non vide then
9     f ← file vide
10    f.enfiler(T)
11    while f non vide do
12      T_courant = f.defiler()
13      Ajouter racine de T_courant à result
14      if T_courant.gauche non vide then
15        | f.enfiler(T_courant.gauche)
16      end
17      if T_courant.droite non vide then
18        | f.enfiler(T_courant.droite)
19      end
20    end
21  end
22  Renvoyer result
23 end

```

Dans quel ordre a été parcouru cet arbre ?

### Remarque

- Cet algorithme utilise une file FIFO
- Cet algorithme n'est pas récursif.

### Exercice 8.112

Créer une fonction **parcours\_largeur(a)** qui effectue un parcours en largeur d'abord de l'arbre binaire a