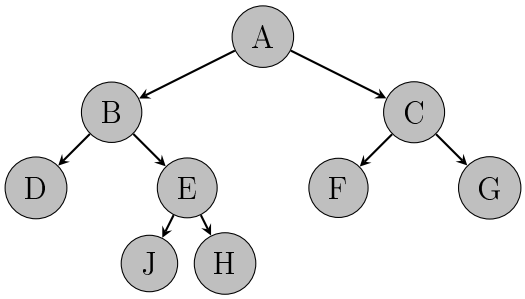
Parcourir un arbre

Parcourir un arbre c'est parcourir tous ses nœuds.Mais dans quel ordre ? Voici les différentes options présentées avec un exemple.



Parcourir mais pour quoi faire ? On supposera que l'on veut effectuer un certain **traitement** sur chaque nœud. Par exemple afficher le nœud ou lire sa valeur.

Parcours en largeur

C'est le parcours dans l'ordre de lecture : A, B, C, D, E, F, G, J, H. Autrement dit, on parcours par génération : l'arrière grand-parent en premier, les grands-parents ensuite, puis les parents et enfin les enfants. Pour obtenir ce résultat, on part de la racine et on utilise une **file**.

```
Entrées : arbre
début
    créer une file vide
    enfiler la racine de l'arbre
    tant que file non vide faire
        défiler un nœud,
        enfiler les enfants du nœud,
        exécuter le traitement sur le
          nœud
    fin
fin
```

préfixe, infixe, postfixe

Dans le parcours en profondeur qui va suivre, quand on analyse un nœud, par exemple B, on parcours la descendance de B avant de parcourir les frères de B. Mais une question demeure : Fait-on le traitement de B...

- ... avant sa descendance – cas **préfixe**
- ... entre ses enfants – il faut qu'il en ait 2 – cas **infixe**
- ... après ses enfants – cas **postfixe**

Selon le cas, on devra opter pour l'un ou l'autre.

Parcours en profondeur préfixe

On parcours la descendance d'un nœud avant de parcourir le frère, mais on fait le traitement sur le parent avant le les enfants. Cela donne donc ici : A, B, D, E, J, H, C, F, G. Dans ce cas, on utilise une **pile**.

```
Entrées : arbre
1 début
2 |  créer une pile vide
3 |  empiler la racine de l'arbre
4 |  tant que pile non vide faire
5 |  |  dépiler un nœud,
6 |  |  empiler les enfants du nœud en ordre
  |  |   inverse,
7 |  |  exécuter le traitement sur le nœud
8 |  fin
9 fin
```

Approche récursive

Le parcours précédent peut être adapté dans pour les cas infixe et postfixe mais on est contraint à quelques complications. Une approche récursive à l'avantage de rester très simple dans tous les cas.
Ci-dessous un cas **infixe**.

```
fonction principale
   Entrées : arbre
1 début
2 |  Appeler fonction
  |   annexe sur la racine
3 fin
```

```
fonction annexe récursive
   Entrées : nœud
1 début
2 |  si nœud a un enfant gauche alors
3 |  |  Appeler fonction annexe avec
  |  |   enfant gauche
4 |  fin
5 |  Effectuer le traitement sur le
  |   nœud,
6 |  si nœud a un enfant droit alors
7 |  |  Appeler fonction annexe avec
  |  |   enfant droit
8 |  fin
9 fin
```

Il n'est pas difficile d'adapter cet algorithme pour les deux autres cas : préfixe et postfixe.