TP numéro 4

Récursivité profonde

• Définir une fonction qui compte en profondeur le nombre de nombres positifs d'une liste quelconque.

```
(nbsup0 '(& -3 2 (1 (2 "cvb") -4) r)) → 3
```

• Écrire une fonction qui calcule en profondeur le nombre d'occurrences d'un élément dans une liste.

```
(occ-prof 'a '(a (b) (c ((a e) f)))) -> 2
```

• Définir une fonction qui remplace en profondeur tout nombre d'une liste quelconque par sa valeur absolue (pensez à utiliser la fonction prédéfinie `abs`).

```
(absliste '(& -3 2 (1 (2 "cvb") -4) r)) → (& 3 2 (1 (2 "cvb") 4) r)
```

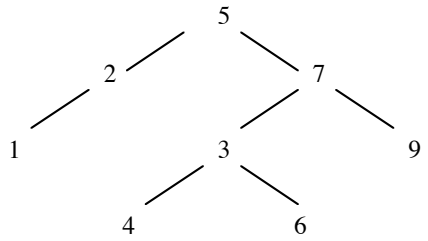
• Écrire une fonction qui calcule la profondeur d'une liste.

```
(profondeur '(a (b c) d)) -> 2
(profondeur '(a (b) (c ((d e) f)))) -> 4
```

Arbres binaires

Commencez par implémenter les fonctions primitives sur les arbres vues en cours :
vide, vide?, valeur, fils-g, fils-d, cons-binaire, arbre=?, feuille ?, arbre ?

Pensez bien à tester vos fonctions sur un arbre où au moins un nœud a un seul fils en définissant par exemple l'arbre `a` suivant :



• Écrire une fonction qui compte le nombre de nœuds d'un arbre.

```
(nb-noeuds a) → 8
```

• Écrire une fonction qui compte le nombre de feuilles d'un arbre.

```
(nb-feuilles a) → 4
```

• Écrire une fonction qui multiplie par deux les valeurs des nœuds d'un arbre de nombres.

```
(fois2 a) → (10(4(2()())())(14(6(8()())(12()()))(18()()))))
```

• Écrire une fonction qui renvoie la liste des valeurs des feuilles d'un arbre.

```
(feuilles a) -> (1 4 6 9)
```

• Écrire une fonction qui prend un arbre binaire et renvoie son miroir : pour tous les nœuds de l'arbre, le fils gauche devient le fils droit et le fils droit devient le fils gauche.

```
(miroir a) -> (5(7(9()())(3(6()())(4()())))(2()(1()())))
```