TP numéro 4

Récursivité profonde

• Définir une fonction qui compte en profondeur le nombre de listes d'une liste quelconque (y compris elle-même).

```
(nbl '(3 (y k) (8 (5 "ert") t) d)) → 4
```

• 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
```

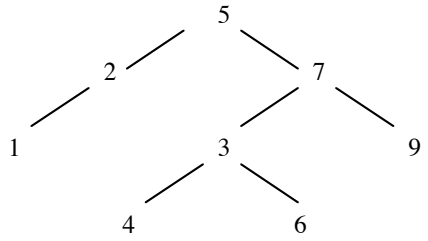
• Définir une fonction qui remplace en profondeur tout nombre d'une liste quelconque par sa valeur absolue (pensez à utiliser la fonction prédéfinie `abs`).

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

Arbres binaires

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

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()())))
```

Listes de listes : une autre version du triangle de Pascal (cf. TD6)

• Écrire une fonction `som2` qui, étant donnée une liste de nombres, retourne la liste des sommes des éléments consécutifs de cette liste.

```
(som2 '(1 2 3 4)) -> (3 5 7)
```

• Utiliser la fonction `som2` pour écrire une autre version de la fonction qui construit une liste de listes correspondant aux n premières lignes du triangle de Pascal.

• Que pensez-vous de la complexité de cette version de la fonction par rapport à celle écrite en TD ?