

## UE ALGO5 — TD2 — Séance 8 : Arbres n-aires

### Objectifs

À la fin de cette séance, vous devriez être capable de :

- manipuler et concevoir des arbres n-aires comme des structures abstraites ;
- réfléchir aux propriétés des arbres n-aires ;
- proposer des implémentations d'arbres n-aires cohérentes avec les spécifications choisies en utilisant des structures sous-jacentes adaptées.

On s'intéresse ici à l'implémentation d'un type abstrait «Arbre n-aire», en utilisant le type «Arbre binaire» vu lors de la séance précédente.

Un arbre n-aire est ici implémenté comme suit :

- le «fils aîné» d'un nœud n-aire est implémenté par le «fils gauche» du nœud binaire du père ;
- le «frère cadet» d'un nœud n-aire est implémenté par le «fils droit» du nœud binaire du frère précédent.

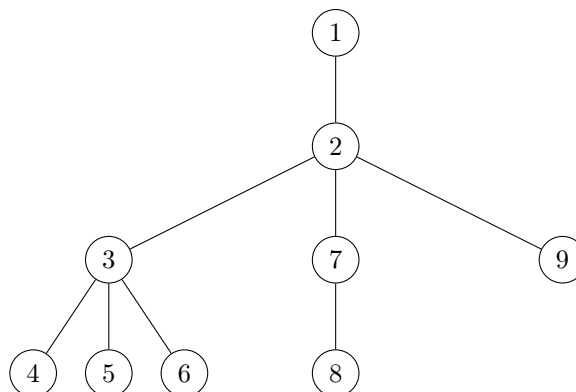
### Introduction

Il existe différentes manières d'implémenter un arbre n-aire. Le principe repose sur le fait qu'un arbre est un couple (élément racine, liste d'arbres). Cette liste d'arbres peut être implémentée comme n'importe quelle liste : représentation contiguë dans un tableau, liste chaînée, etc. Dans la suite du TD on choisit de représenter une liste par un arbre.

Il s'agit ici de montrer qu'une même structure (un arbre binaire) peut implémenter en même temps plusieurs types abstraits : ici, des arbres n-aires et des listes d'arbres, les deux types étant implémentés par un arbre binaire. L'exercice consiste à ne pas construire une liste d'arbres, mais d'utiliser la structure sous-jacente, avec des primitives de manipulation différentes selon le type abstrait.

### Exercice 1.

Q1. Traduisez l'arbre ci-dessous dans sa représentation binaire :

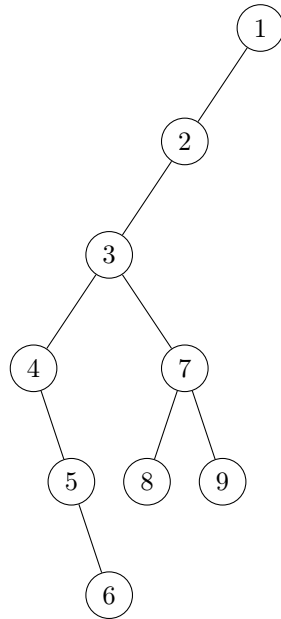


Nous aurons également besoin d'une notion de liste d'arbres : ici nous allons **également** représenter une liste par un arbre binaire : le premier élément de la liste se trouve à la racine de l'arbre, et les éléments suivants sont situés dans son fils droit (qui est inutilisé pour l'instant, regardez votre arbre de la question précédente pour vous en convaincre).

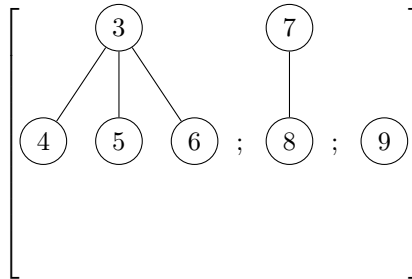
Il peut sembler confus d'utiliser la même structure sous-jacente (les arbres binaires) pour implémenter deux types différents, mais il faut se souvenir qu'on utilisera des primitives de manipulation différentes

pour les listes et pour les arbres  $n$ -aires, il n'y aura donc pas d'ambiguïté.

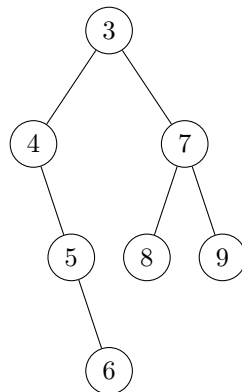
Corrigé —



Q2. Traduisez la liste d'arbres ci-dessous dans sa représentation binaire :



Corrigé —



## Exercice 2.

Q2. Implémentez les primitives du type abstrait «arbre  $n$ -aire», telles que définies ci-dessous.

<sub>1</sub> Élément : un **type**

Arbre : un **type**

3 ListeArbre : un **type**

5 ArbreVide  
 { **Données** : *aucun*  
 7 **Résultat** : *un Arbre vide* }

9 NouveauNoeud  
 { **Données** : *un Élément x, une ListeArbre L*  
 11 **Résultat** : *un Arbre constitué du nœud x, dont les fils sont les éléments de L*  
**Effet de bord** : *un nouveau nœud a été créé* }

13 EstArbreVide  
 15 { **Données** : *un Arbre A*  
**Résultat** : *un booléen vrai ssi A est un Arbre vide* }

17 Elem  
 19 { **Données** : *un Arbre A*  
**Résultat** : *l'Élément associé à la racine de A*  
 21 **Pré-condition** : *A est non vide* }

23 ListeFils  
 { **Données** : *un Arbre A*  
 25 **Résultat** : *une ListeArbre*  
**description** : *A doit être non vide, renvoie la liste des fils associée à la racine de A* }

27 { *Manipulation des listes d'arbres* }

29 ListeVide  
 31 { **Données** : *aucun*  
**Résultat** : *une ListeArbre vide* }

33 Cons  
 35 { **Données** : *un Arbre A, une ListeArbre L*  
**Résultat** : *une ListeArbre constituée de l'arbre A, suivi de la liste L.* }

37 EstListeVide  
 39 { **Données** : *une ListeArbre L*  
**Résultat** : *un booléen vrai ssi L est vide.* }

41 Premier  
 43 { **Données** : *une ListeArbre L*  
**Résultat** : *un Arbre, renvoie le premier arbre de la liste L*  
 45 **Pré-condition** : *L non vide* }

47 Suivants  
 { **Données** : *une ListeArbre L*  
 49 **Résultat** : *une ListeArbre, renvoie la liste des arbres suivants le premier.*  
**Pré-condition** : *L non vide* }

---

## Corrigé —

Implémentation : le plus difficile à comprendre est Premier(L), qui retourne L : on peut s'aider d'un schéma montrant que l'arbre binaire représentant la liste d'arbres L et son premier élément sont bien les mêmes!  
 Explication : le type change, et par conséquent l'ensemble des primitives applicables sur L. Si L est une liste d'arbres, on peut appliquer la primitive Suivant et récupérer le fils droit de l'arbre binaire sous-jacent. Si par contre L est un arbre, aucune primitive ne permet de partir sur ce fils droit...

Cela dit, on aurait aussi pu renvoyer NouveauNoeudBin(FGauche(L),Elem(L),ArbreBinVide).

```
Arbre : le type ArbreBin
2 ListeArbre : le type ArbreBin

4 ArbreVide:
    retourner ArbreBinVide
6
NouveauNoeud(x,L):
8    retourner NouveauNoeudBin(L,x,ArbreBinVide)

10 EstArbreVide(A):
    retourner EstArbreBinVide(A)
12
Elem(A):
14    retourner ElemBin(A)

16 ListeFils(A):
    retourner FGauche(A)
18
ListeVide:
20    retourner ArbreBinVide

22 Cons(A,L):
    retourner NouveauNoeudBin(FGauche(A),Elem(A),L)
24
EstListeVide(L):
26    retourner EstArbreBinVide(L)

28 Premier(L):
    retourner L
30
Suivants(L):
32    retourner FDroit(L)
```

---