TP 8 : Arbres en OCaml

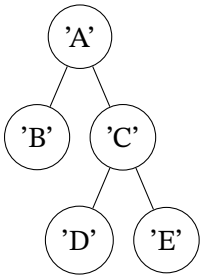
On commence par manipuler simplement des arbres en OCaml. On va commencer par les arbres binaires non stricts, puis on pourra passer aux arbres binaires stricts. Ici, les ★ veulent simplement dire que c'est plus difficile / qu'on n'a pas encore forcément touché au cours nécessaire pour répondre.

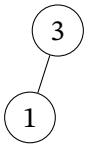
Exercice 1 – Arbres binaires non stricts en OCaml

On va utiliser le type suivant :

```OCaml
type 'a arbre =
  | Vide
  | Noeud of 'a * 'a arbre * 'a arbre
```

▶ **Question 1** Représenter les arbres suivants avec ce type :





- un arbre vide,
- un arbre de hauteur 3 à 4 nœuds,
- un arbre de hauteur 3 à 15 nœuds (réfléchir une seconde avant de faire un exemple à 15 lignes),
- ★ un arbre de hauteur 11 à 4095 nœuds (réfléchir une seconde avant de faire un exemple à 4095 lignes),

▶ **Question 2** Tester vos fonctions `hauteur` `nombre_noeuds` `liste_sous_arbres` sur ces exemples et vérifier que le résultat correspond à vos attentes.

▶ **Question 3** Écrire une fonction `est_feuille : 'a arbre -> bool` vérifiant si un noeud est une feuille. *On s'interdira d'utiliser cette fonction dans la suite, mais on notera et on chérira la beauté du motif que l'on y utilise.*

▶ **Question 4** Écrire une fonction `nb_feuilles : 'a arbre -> int` qui renvoie le nombre de feuilles d'un arbre.

▶ **Question 5** Écrire une fonction `somme_etiquettes : int arbre -> int` qui renvoie la somme des étiquettes des nœuds de l'arbre.

▶ **Question 6** Écrire une fonction `max_etiquettes : 'a arbre -> 'a` qui renvoie le maximum des étiquettes des nœuds de l'arbre. On lèvera une exception si l'arbre est vide.

▶ **Question 7** Écrire une fonction `somme_feuilles : int arbre -> int` qui renvoie la somme des étiquettes de toutes les feuilles.

▶ **Question 8** Écrire une fonction `parcours : 'a arbre -> 'a list` qui renvoie la liste des étiquettes d'un arbre, dans l'ordre que vous voulez, mais une et une seule fois chacune. On pourra utiliser l'opérateur `@`.

▶ **Question 9** Écrire une fonction `max_min_arbre : int arbre -> int * int` qui renvoie le minimum et le maximum des étiquettes d'un arbre. Chaque nœud ne devra être parcouru qu'une seule fois. On lèvera une exception si l'arbre est vide. Attention : il ne doit y avoir, lors d'un appel de la fonction, qu'au plus un appel récursif sur chaque fils.

▶ **Question 10** Une branche est un chemin de la racine vers une feuille. Le poids d'une branche est la somme des étiquettes des nœuds qui la composent. Écrire une fonction `max_somme_branche : int arbre -> int` qui renvoie le poids de la branche de poids maximal.

On considère maintenant le type :

```OCaml
type 'a arbre_strict =
  | Feuille of 'a
  | Noeud_Interne of 'a arbre_strict * 'a * 'a arbre_strict
```

On remarque qu'il n'y a plus la possibilité de définir un arbre vide. Une feuille est maintenant un arbre qui est une `Feuille`. La taille d'un arbre est le nombre total de nœuds (nœuds internes et feuilles). Les autres définitions sont inchangées.

▶ **Question 11** Certains des arbres définis à la section précédente ne peuvent plus l'être avec ce nouveau type. Lesquels ? Pourquoi ?

▶ **Question 12** Reprendre toutes les questions de la section précédente avec ce nouveau type.

★ **Question 13** *Quand on aura parlé de parcours d'arbres*, implémenter les parcours d'arbres pour ce nouveau type.

Exercice 2 – Arbres généraux et forêts

Pour ne pas m'embêter, je réutilise le nom de type `arbre` *ici. Pour être plus propre, je vous conseille de commencer un nouveau fichier ici.*

Dans cet exercice, on s'intéresse aux nœuds d'arité quelconque, avec un point de vue légèrement différent du cours. Pour cela, on introduit la notion de *forêt* :

Définition 1

Une *forêt* est un ensemble fini (éventuellement vide) d'arbres non vides sans nœuds communs.

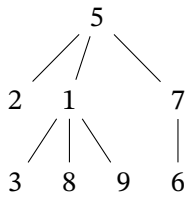
On peut donc voir les fils d'un nœud d'un arbre général comme une forêt. En OCaml on peut aussi définir les arbres généraux ainsi :

OCaml
```
type 'a arbre = Vide | Noeud of 'a * 'a arbre list;;
```

On peut vouloir donner un nom plus explicite à une liste d'arbres qui est une forêt ! Mais alors pour définir une forêt, il faut savoir définir un arbre et réciproquement...De la même manière que l'on peut définir des fonctions mutuellement récursives, on peut définir des types mutuellement récursifs :

OCaml
```
type 'a arbre = Vide | Noeud of 'a * 'a foret
  and 'a foret = 'a arbre list
(* comme au DS ! *)
```

▶ **Question 1** Implémenter l'arbre suivant :



Les notions de taille et profondeur existent toujours sur les arbres généraux. On peut les programmer en OCaml à l'aide de deux fonctions mutuellement récursives :

OCaml
```
let rec taille arbre =
  match arbre with
  | Vide -> 0
  | Noeud (_, fils) -> 1 + taille_foret fils
and taille_foret foret =
  match foret with
  | [] -> 0
  | arbre :: autres -> (taille arbre) + (taille_foret autres)
```

▶ **Question 2** Écrire la fonction `hauteur : 'a arbre -> int`. On pourra définir la fonction mutuellement récursive `max_hauteur_foret : 'a arbre list -> int`.

▶ **Question 3** Écrire une fonction `degre : 'a arbre -> int` qui donne le degré d'un arbre, c'est-à-dire le plus grand degré de ses nœuds.

Si on suppose que l'on n'a pas besoin de l'arbre vide (ce qui est assez naturel dans certaines études) on peut se passer du constructeur de type. Pour bien insister sur le fait que l'on manipule un arbre et non un couple quelconque, on propose le type suivant :

```OCaml
type 'a arbre_general = {etiquette : 'a; fils : 'a arbre_general list}
```

On aurait pu aussi utiliser un type somme avec un unique constructeur paramétré, comme on l'a mentionné dans le cours.

▶ **Question 4** Réécrire les fonctions `taille` et `hauteur` avec ce nouveau type.

▶ **Question 5** Peut-on écrire une fonction qui renverse l'ordre des fils de chaque nœud d'un arbre (du *même* arbre, sans en recréer un nouveau) ? Pourquoi ?

▶ **Question 6** Écrire une fonction `nb_feuilles : 'a arbre_general -> int` qui renvoie le nombre de feuilles d'un arbre.

▶ **Question 7** Écrire une fonction `max_etiquettes : 'a arbre_general -> 'a` qui renvoie le maximum des étiquettes des nœuds de l'arbre.

▶ **Question 8** Adapter la fonction `max_somme_branche : int arbre_general -> int` qui renvoie le poids de la branche de poids maximal. *Une branche est un chemin allant de la racine jusqu'à une feuille.*