TD (feuille 6) : Arbres

Exercice 1 - Une autre représentation des arbres

En cours, vous avez vu la définition d'arbres binaires polymorphes suivante :

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

1. Définir le type `('a,'b) arbre` des arbres binaires polymorphes où une information de type `'b` est associée aux feuilles.
2. Définir les fonctions `taille: ('a,'b) arbre -> int`, `profondeur: ('a,'b) arbre -> int` et `est_complet: ('a,'b) arbre -> bool`.
3. Définir la fonction `nb_feuilles: ('a,'b) arbre -> int` qui compte le nombre de feuilles d'un arbre.
4. Définir la fonction `feuilles: ('a,'b) arbre -> 'b list` qui retourne la liste des feuilles d'un arbre.
5. Définir la fonction `map n f a` qui applique la fonction `n` aux noeuds de `a` et la fonction `f` à ses feuilles.
   `val map : ('a -> 'b) -> ('c -> 'd) -> ('a, 'c) arbre -> ('b, 'd) arbre`

Exercice 2 - Un peu d'arithmétique

On veut représenter des expressions arithmétiques contenant les opérateurs binaires $+$, $-$, $\times$, $/$, ainsi que l'opérateur unaire $-$. On veux aussi manipuler des constantes (1, 2, etc.) et des variables identifiées par un caractère (a, b, etc.).

1. Donner la représentation sous forme d'arbre des expressions suivantes :
   (a) $-(a \times b)$
   (b) $(a + b) \times c$
   (c) $-(a + b) \times -a \times -b$
2. Définir le type `expr_arith` des expressions arithmétiques.
3. Donner les valeurs OCaml correspondant aux expressions précédentes.
4. Définir une fonction `simplifier: expr_arith -> expr_arith` qui effectue les simplifications suivantes :
   (a) $- - a \rightarrow a$
   (b) $1 \times a \rightarrow a$ et $a \times 1 \rightarrow a$
   (c) $0 \times a \rightarrow 0$ et $a \times 0 \rightarrow 0$
   (d) $0 + a \rightarrow a$ et $a + 0 \rightarrow a$
5. Définir la fonction `eval env e` qui évalue l'expression `e` dans un environnement `env` qui à chaque constante identifiée par un caractère associe une valeur entière.
   `val eval: (char * int) list -> expr_arith -> int`