TP3 *Types Sommes*

Exercice 1 : Arbres binaires polymorphes (pour s'échauffer)

1. Définir un type arbre binaire dont seules les feuilles sont étiquetées. Un arbre binaire est soit une feuille étiquetée par une valeur de type α, soit un noeud dont les deux fils sont eux-même des arbres de même type.
2. Définir un arbre de ce type, comportant au moins 4 feuilles.
3. Écrire une fonction qui compte le nombre de nœuds internes d'un arbre.
4. Écrire une fonction qui compte le nombre de feuilles d'un arbre.
5. Écrire une fonction qui retourne la profondeur d'un arbre (longueur de la plus longue branche).
6. Écrire une fonction qui teste si deux arbres ont la même forme (sans tenir compte des valeurs des feuilles).
7. Écrire une fonction qui retourne la liste des valeurs des feuilles d'un arbre.
8. Écrire une fonction `map_arbre` qui applique une fonction à toutes les étiquettes d'un arbre. Puis utiliser cette fonction pour définir une fonction qui permet d'incrémenter toutes les feuilles d'un arbre d'entiers.

Exercice 2 : Expressions arithmétiques avec variables

On désire représenter des expressions arithmétiques avec variables. On autorise seulement deux opérateurs binaires : la multiplication, et l'addition, et un opérateur unaire : le moins. Une expression sera donc une constante numérique, ou une variable, ou un opérateur unaire appliqué à une expression, ou un opérateur binaire appliqué à 2 expressions. On définit les types suivants permettant de représenter ces expressions par des arbres :

```
type operateur_bin = Mult | Add;;
type operateur_un = Moins;;
type arbre = Const of int
           | Var of string
           | Noeud1 of (operateur_un * arbre)
           | Noeud2 of (operateur_bin * arbre * arbre);;
```

la constante numérique 3	sera représentée par	`Const 3`
la variable "x"	sera représentée par	`Var "x"`
l'expression -x	sera représentée par	`Noeud1(Moins, Var "x")`
l'expression x*3	sera représentée par	`Noeud2(Mult,Var "x", Const 3)`

1. Écrire une fonction (`chaine_de_arbre e`) qui convertit un arbre en chaîne de caractère entièrement parenthésée. Ex : `"((x*3)+y)"`. On pourra utiliser la fonction `string _of_int : int → string` qui convertit un entier en une chaîne de caractères.
2. On donne des valeurs pour les variables sous forme d'une liste d'association dont chaque élément est un couple (`nom, val`) où `nom` est une chaîne de caractères (nom de la variable) et `val` est un numérique (valeur de la variable). On dit qu'une expression e est *close* relativement à une liste d'association L si toutes les variables qui apparaissent dans e ont une valeur dans L. Écrire une fonction (`close e L`) qui teste si une expression est close.
3. Écrire une fonction (`eval e L`) qui calcule la valeur de l'expression e.

Exercice 3 : Arbres n-aires

On représente encore des expressions arithmétiques, mais en utilisant cette fois des opérateurs n-aires. On autorise 3 opérateurs n-aires : la multiplication, l'addition et la soustraction. Une expression sera une constante numérique, ou un opérateur n-aire appliqué à une liste (non vide) d'expressions.
On définit les types suivants permettant de représenter ces expressions par des arbres :

```
type operateur = Mult | Plus | Moins;;
type arbre = C of int
           | N of (operateur * arbre list);;
```

la constante numérique 3 sera représentée par	`C 3`
l'expression (1 + 4 + 7) sera représentée par	`N(Plus, [C 1; C 4; C 7])`
l'expression (1 +(5 * 2))sera représentée par	`N(Plus, [C 1; N(Mult,[C 5; C 2]])`

Pour chacune des fonctions à écrire, utiliser si possible it_list ou list_it, et donner des versions avec et sans map.

1. Écrire une fonction qui retourne le nombre de constantes d'un arbre donné en argument.
2. On dit qu'un expression est correcte si tous les opérateurs qu'elle contient sont appliqués à des listes non vide d'arguments. Écrire une fonction qui teste si un arbre représente une expression correcte.
3. Écrire une fonction qui calcule la valeur d'un arbre.
4. Écrire une fonction `(chaine_de_arbre e)` qui convertit un arbre en chaîne de caractère entièrement parenthésée. Ex : `"(1+4+7)"` ou `"(1+(5*2))"`.