TP noté – mardi 5 avril 2016

Règles : en début de TP, recopiez les fonctions primitives sur les arbres du TP 4 ou 5. Pour la suite de la séance, aucun document ni accès au Web ni consultation de vos précédents TPs n'est autorisé. Le seul document autorisé est l'aide-mémoire.

Contenu de votre fichier de programme : votre fichier doit contenir en première ligne vos nom, prénom et numéro d'étudiant. Les commentaires doivent au minimum concerner les types des arguments et des résultats de vos fonctions, ainsi que les tests réalisés, avec leurs résultats. Cela compte pour une part importante de l'évaluation. N'hésitez pas à ajouter des hypothèses si un exercice ne vous paraît pas suffisamment spécifié.

Procédure de rendu du TP : en fin de séance, vous devez déposer votre fichier sur TOMUSS dans la colonne « `TPnote_Rendu` ».

Pour l'ensemble des exercices, l'utilisation des fonctions `map` et `apply` n'est pas autorisée.

Exercice 1

Définir une fonction booléenne qui teste si une liste plate de nombres est constituée uniquement de nombres strictement positifs.

```
(liste-nb-positif? '(1 7 8 4))  → #true
(liste-nb-positif? '(9 7 -1 2)) → #false
(liste-nb-positif? '(9 0 1 2))  → #false
```

Exercice 2

Définir une fonction qui duplique les nombres et supprime les éléments qui ne sont pas des nombres d'une liste plate.

```
(modif-liste '(1 2 3 4))        → (1 1 2 2 3 3 4 4)
(modif-liste '(a b c d))        → ()
(modif-liste '(1 a b 3 d))      → (1 1 3 3)
```

Exercice 3

Définir une fonction qui enlève les n premiers éléments de type liste d'une liste quelconque.

```
(enleve 3 '(a b c))                    → (a b c)
(enleve 2 '((a) 1 2 (r) (e f) ))       → (1 2 (e f))
(enleve 2 '( ((a)(b c)) 1 2 (r) (e f) )) → (1 2 (e f))
```

Exercice 4

Définir une fonction qui, étant donnée une liste de nombres, renvoie une liste composée du nombre d'éléments pairs et du nombre d'éléments impairs de cette liste. Attention à ne parcourir la liste qu'une seule fois.

1- Ecrire une première version de cette fonction pour effectuer les calculs en remontant.

2- Ecrire une seconde version de cette fonction pour effectuer les calculs en descendant.

```
(pair-impair '(1 6 7 4 5)) → (2 3)
```

Exercice 5

Définir une fonction qui, étant donnée une liste de nombres, remplace en profondeur les nombres de la liste par un booléen spécifiant si le nombre correspondant est pair ou non.

```
(pair? '(2 (3) 5 (6 (7)))) → (#t (#f) #f (#t (#f)))
```

Exercice 6

Définir une fonction qui, étant donné un arbre binaire, compte le nombre de nœuds de cet arbre dont la valeur est de type nombre.

```
(nombre-noeuds-nb '(5 (4 ()(c ())())) (3 (2 ()())()))) → 4
```

Exercice 7

Définir une fonction qui prend en paramètre un arbre binaire et qui retourne un arbre dont les valeurs des nœuds correspondent à une liste de deux valeurs : la première valeur correspond au nombre de nœuds ayant une valeur de type « nombre » et la seconde au nombre de nœuds dont la valeur n'est pas de type « nombre » dans les descendants du nœud considéré ainsi que lui-même.

```
(arbre-nb-symb '(12 (1 (c ())())())(5 ()()))) →
                ((3 1) ((1 1) ((0 1) () ()) ()) ((1 0) () ()))
```