TP 2 : Types, Listes

Exercice 1 - Type enuméré

1.1 Définir un type enuméré `euro` pour représenter l'ensemble des pièces (1c, 2c, 5c, 10c, 20c, 50c, 1€, 2€) et billets (5€, 10€, 20€, 50€, 100€, 200€, 500€) de l'euro.

1.2 Écrire une fonction `est_billet: euro -> bool` qui retourne `true` si l'argument passé correspond à un billet, et `false` sinon.

1.3 Écrire une fonction `montant: euro list -> int` calculant le montant (en centimes) correspondant à la liste de pièces/billets passée en argument.

1.4 Écrire une fonction `monnaie: int -> euro list` qui, sur la donnée d'une somme en centimes, renvoie une liste de pièces/billets la plus courte possible correspondant à cette somme.

Exercice 2 - Type union

On considère le type suivant :

```
type nombre = Int of int | Real of float ;;
```

2.1 Écrire une fonction `division` qui prend en argument deux éléments de type `nombre` et qui calcule le quotient de ces deux nombres. On utilisera le type `int` autant que possible.

2.2 Écrire une fonction `compare_nombre` qui prend en argument deux éléments de type `nombre` et qui renvoie `-1` si le premier nombre est strictement plus petit que le deuxième, `0` si les deux nombes sont égaux, et `1` sinon.

Exercice 3 - Les listes

3.1 Écrire une fonction récursive `produit` qui, sur la donnée d'une liste d'entiers, renvoie le produit de tous les éléments de cette liste.

3.2 Écrire une fonction récursive `carre_liste` qui, sur la donnée d'une liste d'entiers `l`, renvoie la liste des carrés des éléments de `l`.

3.3 Écrire une fonction récursive `min_max` qui, sur la donnée d'une liste d'entiers, renvoie le couple formé du plus petit et du plus grand entier dans cette liste.

note : Pour la liste vide, on pourra renvoyer `(max_int, min_int)`. Pourquoi ?

3.4 Écrire une fonction récursive `recherche` qui, sur la donnée d'un élément `e` et d'une liste `l`, renvoie `true` si `e` apparaît dans `l`, et `false` sinon.

3.5 Écrire une fonction récursive `nb_occurrence` qui, sur la donnée d'un élément `e` et d'une liste `l`, renvoie le nombre de fois où `e` apparaît dans `l`.

3.6 Écrire une fonction récursive `nub` qui, sur la donnée d'une liste `l`, renvoie la liste sans les doublons. Par exemple, `nub [1;1;2;3;1;4]` devra retourner `[1;2;3;4]` ou `[2;3;1;4]` (selon l'algorithme utilisé).

3.7 Refaire les questions précédentes sans utiliser de fonctions récursives. Utiliser pour cela les fonctions du module `List`, notamment `List.map` et `List.fold_left`/`List.fold_right`.