Fiche de TP no. 2

Conditionnels, conditions de garde, et filtrage

Exercice 1. Définissez, dans une script, deux versions de la fonction `estVide`. Cette fonction prend en argument une liste et retourne un Booléen selon que son argument est la liste est vide ou non. Dans la première versions on utilisera les conditionnels, dans le deuxième le filtrage. Écrivez, avant chaque définition, le type de la fonction définie. Quel est la version de la fonction `estVide` dont le typage est plus général ?

Exercice 2. Considérez la fonction `safetail` qui se comporte exactement comme `tail`, sauf qu'elle envoie la liste vide vers elle même (rappel : `tail []` produit un erreur).
Définissez, dans un script, trois versions de la fonction `safetail` en utilisant :

(a) une expression conditionnelle ;

(b) des équations avec conditions de garde ;

(c) le filtrage par motifs.

Exercice 3. Considérez le script suivant :

```
import Data.Char

areAlpha :: String -> Bool
areAlpha xs = if xs == [] then
                  True
              else if isAlpha (head xs)
                   then
                       areAlpha (tail xs)
                   else False
```

Copiez ce script, et ajoutez trois définition alternatives de la fonction `areAlpha`

1. en utilisant les conditions de garde,
2. en utilisant le filtrage par motifs,
3. en utilisant les fonctions `map` et `and`.

Compréhension sur les listes

Exercice 4. Le produit scalaire de deux listes d'entiers xs et ys de longueur n est la somme des produits des entiers correspondants :

$$xs \cdot ys = \sum_{i=0}^{n-1} xs_i * ys_i \,.$$

1. Utilisez la compréhension et les fonctions vues en cours pour définir une fonction qui retourne le produit scalaire de deux listes.
2. Autrement, utilisez l'induction pour définir cette fonction.

Exercice 5. Un nombre positif est *parfait* s'il est la somme de tous ses facteurs, sauf lui même. En utilisant la compréhension, définissez une fonction

```
perfects :: Int -> [Int]
```

qui retourne la liste de tous les nombres parfaits, jusqu'à la limite passée en paramètre. (Utilisez la fonction `factors` vue en cours).

Expressions lambda

Exercice 6. Considérez le script suivant :

```
carre x = x^2

moyenne ns = sum ns/fromIntegral (length ns)

norme ns =  sqrt (moyenne (map carre ns))

main = print (norme [1..5])
```

1. Ajoutez, dans le script, la déclaration du type avant chaque fonction définie.
2. En utilisant la notation lambda, éliminez du script toute définition intermédiaire (`carre`, `moyenne`, `norme`) en définissant ainsi le `main` dans une seule expression.

Types et classes

Exercice 7. Considérez l'algorithme `qsort` présenté en premier cours :

```
qsort [] = []
qsort (x:xs) =
    qsort smaller ++ [x] ++ qsort larger
    where
      smaller = [a | a <- xs, a < x]
      larger  = [b | b <- xs, x < b]
```

1. Copiez cet algorithme dans un script, qui sera chargé avec `ghci`.
2. Tapez `:type qsort` dans `ghci` et expliquez la réponse donnée.
3. Tapez `:info Ord` dans `ghci` et decodez la réponse donnée.

Si vous avez complété tous les exercices, en voici un autre

Exercice 8. Un triplet (x, y, z) d'entiers positifs est dit *de Pythagore* si $x^2 + y^2 = z^2$.

1. Définissez, en utilisant la compréhension, une fonction

   ```
   pyths :: Int -> [(Int,Int,Int)]
   ```

   qui envoie un entier n vers la listes de tous les triplets de Pythagore avec composantes dans `[1..n]`.
2. Si (x, y, z) est un triplet de Pythagore, alors (y, x, z) est aussi un triplet de Pythagore. Proposez une solution à l'exercice précèdent, où seulement un triplet parmi (x, y, z) et (y, x, z) apparaît dans la liste retournée.
3. Si (x, y, z) est un triplet de Pythagore, alors $x, y < z$. Proposez un solution de l'exercice précèdent qui utilise cette observation pour accélérer les calculs.