Fiche de TP no. 2

Caveats : *le temps passe vite ! Si, après vingt minutes, vous êtes encore au premier exercice, cela est un bon indice que vous ne vous donnez pas les moyens d'apprendre.*
Objectifs : *Comprendre les contraintes de classe. Maîtriser la syntaxe des conditionnels, les définitions par équations avec conditions de garde et celle par filtrage ; savoir distinguer parmi ces constructions. Se familiariser avec les fonctions anonymes et les expressions de liste. Utilisez 1h20 de votre temps pour résoudre les exercice 1 à 6, puis passez aux exercices sur les listes (déjà vus en TD).*

Types et classes

Exercice 1 : *Informations sur types et classes dans `ghci`.* 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.

Conditionnels, équations avec conditions de garde, et filtrage

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. Dans le script suivant on définit une fonction qui calcule la longueur du plus grand préfixe d'une chaîne de caractères composé par des lettres seulement :

```
import Data.Char(isAlpha)

lengthPrefixAlpha :: String -> Int
lengthPrefixAlpha xs = if xs == [] then 0 else
                         if isAlpha (head xs) then
                           1 + lengthPrefixAlpha (tail xs)
                         else 0
```

Copiez ce script, et ajoutez trois définitions alternatives de la même fonction

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

Expressions lambda

Exercice 4 : *Se familiariser avec les expressions lambda.* 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.

Exercice 5. Considérez les définitions des fonctions `power` et `distMoy` suivantes :

```
power 0 = \_ -> \x -> x
power n = \f -> \x -> f (power (n-1) f x)
distMoy = \xs ys -> (\zs-> sum zs / fromIntegral (length zs))
                    (map (\(x,y) -> abs (x - y)) (zip xs ys))
```

En utilisant `ghci`

1. donnez un type à ces fonctions, et à toute expression lambda intermédiaire ;
2. testez ces deux fonctions, et donnez une description ce ce qu'ils font en général ;
3. réécrivez ces deux définitions en utilisant une notation qui vous est (peut être) plus habituelle ; par exemple :
   — déplacez, dans la définition d'une fonction, les paramètres formels de droite à gauche du l'égalité ;
   — introduisez des définitions (et des noms) intermédiaires pour les fonctions anonymes.

Compréhension sur les listes

Vous avez partiellement résolu ces exercices en TD : il s'agit maintenant de les transformer en des script Haskell et de les tester.

Exercice 6. 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 7. 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).

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 une solution de l'exercice précèdent qui utilise cette observation pour accélérer les calculs.