Programmation Fonctionnelle L2 – Examen Final 26/5/2021

Consignes

- **Documents autorisés : une feuille manuscrite A4 recto-verso.**
- Vous avez le droit d'utiliser toutes les fonctions de la bibliothèque standard de Haskell, mais pas des fonctions qui se trouvent dans d'autres modules. La seule exception est la bibliothèque `System.IO`, que vous pouvez utiliser dans l'exercice 4.
- Pour les exercices de programmation, vous avez le droit de programmer des fonctions auxiliaires si vous en avez besoin.
- Vous devez préciser les types de vos fonctions.

1 Types (2 points)

On considère les fonctions `f1,f2,f3,f4` ci-dessous. Déterminer le type inféré par le compilateur pour chacune de ces fonctions (ou si il y a une erreur de compilation, expliquer quelle est l'erreur).

```
1 f1 (Just x) = [x,x]
2 f2 f x = fmap (map f) x
3 f3 x = map x x
4 f4 xs = do
5  x <- xs
6  Just x
```

2 Que font ces lignes ? (2 points)

Pour cet exercice on considère que les deux fonctions suivantes ont été définies :

```
1 f1 x = if x>0 then Just (x-1) else Nothing
2 f2 x = [x..2*x]
```

Quel est le résultat de l'évaluation de chacune des lignes suivantes ? (Si il y a une erreur de compilation, expliquer quelle est l'erreur)

```
1 fmap (fmap (+2)) [Just 1, Just 2, Nothing]
2 map (++) ["ab","cd"] <*> ["e","f"]
3 return 3 >>= f1 >>= f1
4 [2] >>= f2 >>= f2
```

3 Compléter le programme (3 points)

Pour cet exercice on va programmer deux fonctions : `prefixSum` et `suffixSum`. La fonction `prefixSum` prend comme argument une liste d'entiers `xs` et retourne une liste qui contient la somme de chaque préfixe de `xs`, c'est-à-dire, le n-ième élément de la liste retournée est la somme des n premiers éléments de `xs`. La fonction `suffixSum` fait la même chose pour les suffixes de `xs`. Voici quelques exemples d'utilisation :

```
> prefixSum [1,2,3,4,5]
[1,3,6,10,15]
> suffixSum [1,2,3,4,5]
[15,14,12,9,5]
```

Dans le premier exemple, `prefixSum` retourne `[1, 1+2, 1+2+3, 1+2+3+4, 1+2+3+4+5]`. De la même façon, `suffixSum` retourne `[1+2+3+4+5, 2+3+4+5, 3+4+5, 4+5, 5]`.

1. Dans le programme ci-dessous, remplacer les `(???)` par une expression de Haskell en sorte que `prefixSum` fonctionne comme décrit ci-dessus.
2. Dans le programme ci-dessous, remplacer les `(***)` par une expression de Haskell en sorte que `suffixSum` fonctionne comme décrit ci-dessus.
3. Quelles sont les complexités de vos fonctions pour une liste de taille n, si on suppose que calculer la somme de deux `Integer` a complexité $O(1)$? Pourquoi ?

Programme à compléter :

```
1 prefixSum :: [Integer] -> [Integer]
2 prefixSum xs = foldl (???) [head xs] (tail xs)
3
4 suffixSum :: [Integer] -> [Integer]
5 suffixSum xs = foldr (***) [last xs] (init xs)
```

NB: Il faut compléter le programme donné et pas faire un nouveau programme qui implémente les deux fonctions. Dans tous les cas vous pouvez supposer que la liste donnée n'est pas vide. On vous rappelle que la fonction `last` retourne le dernier élément d'une liste, alors que la fonction `init` retourne une liste qui contient tous les éléments de la liste donnée sauf le dernier.

4 Fichiers (3 points)

Écrire un programme de Haskell qui ouvre un fichier appelé `data.txt`. Ce fichier contient des entiers, séparés par des espaces, éventuellement dans plusieurs lignes. Par exemple, le fichier `data.txt` pourrait être le fichier suivant :

```
1 2
3
4 5 6
```

Vous pouvez supposer que le fichier `data.txt` est valide et suit toujours ce format. Votre programme doit lire les entiers du fichier, calculer leur somme, et puis afficher un message à l'utilisateur. Exemple d'exécution pour le fichier ci-dessus :

```
> runhaskell  file.hs
The sum is: 21
```

5 Tribonacci (3 points)

La suite de Tribonacci est définie de manière similaire à celle de Fibonacci, sauf que chaque terme est la somme des trois termes précédents. Ainsi, si les trois premiers termes sont $0, 1, 1$ la suite de Tribonacci est $0, 1, 1, 2, 4, 7, 13, 24, 44, 81, \ldots$.

Donner une fonction qui, étant donné n, retourne le n-ième terme de la suite de Tribonacci. La complexité de votre fonction devrait être polynomiale en termes de n.

6 AllUnique (4 points)

On a vu la définition suivante pour les arbres binaires.

```
data Tree a = Leaf | Node (Tree a) a (Tree a)
```

1. Quel est le `kind` de `Tree` ?
2. Écrire une fonction `allUnique` qui prend comme paramètre un arbre binaire `t` et retourne `True` si et seulement si `t` ne contient pas de doublons, c'est-à-dire, si il n'existe pas d'élément qui paraît plusieurs fois dans `t`.

On précise que les arbres de cet exercice sont des arbres arbitraires et pas des arbres binaires de recherche (donc, vous ne devrez pas faire des suppositions concernant l'ordre des valeurs contenues dans l'arbre).

7 Collatz (3 points)

On a vu la fonction de Collatz, dont une implémentation en Haskell est la suivante :

```
collatz :: Integer -> Integer
collatz x
 | x `mod` 2 == 0 = x`div`2
 | otherwise = 3*x+1
```

On vous demande d'écrire les fonctions suivantes :

- Une fonction `kTimesCollatz` qui prend comme arguments deux entiers k, x et retourne le résultat qu'on obtient si on applique la fonction `collatz` k fois sur x. Exemple : `kTimesCollatz 3 15` donne 70 car on a $15 \to 46 \to 23 \to 70$.
- Une fonction `invertCollatz` qui étant donné un entier x retourne une liste qui contient (dans n'importe quel ordre) tous les entiers y tels que si on applique `collatz` sur y on obtient x.

  Exemple : `invertCollatz 16` donne `[5,32]` alors que `inverCollatz 15` donne `[30]`.
- Une fonction `kPrevCollatz` qui prend comme arguments deux entiers `k,x` et retourne une liste qui contient (dans n'importe quel ordre) tous les entiers `y` tels que `kTimesCollatz k y` donne `x`.

Autrement dit, la fonction `collatz` qu'on vous donne fait une étape en avant dans le processus de Collatz ; la fonction `kTimesCollatz` fait k étapes en avant, pour un k donné ; la fonction `invertCollatz` fait une étape en arrière ; et la fonction `kPrevCollatz` fait k étapes en arrière, pour un k donné.

Exemples d'utilisation :

```
> kPrevCollatz 3 17
[22,136]
> kPrevCollatz 5 22
[18,112,19,116,117,704]
> map (kTimesCollatz 5) (kPrevCollatz 5 22)
[22,22,22,22,22,22]
```