

# Programmation Fonctionnelle L2 – Examen Partiel 2020

## 1 Types

Considérer le programme suivant.

```

1 f1 'a' = 'b'
2 f1 _ = 'c'
3 f2 [] = [['a']]
4 f3 x | head x == head (tail x) = "a"
5       | otherwise = "b"
6 f4 x y = x y y
7 f5 x = if head x == 'a' then "a" else 0
8 f6 x y = map (x y)
9 f7 x = map x x
10 f8 x = zip x x
11 f9 x y = x ++ f8 y
12 f10 = f6 head
13 f11 x y = map (x+) y
14 f12 x y = zipWith (x+) y

```

- Trouver les lignes qui donnent une erreur de compilation. Expliquer quelle est l'erreur.
- Pour les expressions  $f_1, f_2, \dots, f_{12}$  pour lesquels le programme n'a pas d'erreur de compilation, donner le type inféré par le compilateur.

(4 points)

## 2 Types – Part 2

Pour chacun des types suivants, donner une expression de Haskell qui a ce type, c'est-à-dire, le compilateur infère le type donné pour votre expression.

```

1 String -> String -> String
2 (String -> String) -> String
3 (Char, Char) -> (Char -> Char)
4 Num a => [a] -> [a] -> [a]
5 Ord a => a -> a -> Bool
6 [a] -> [b] -> [a]
7 Bool -> Maybe Char
8 Maybe a

```

(2 points)

### 3 Dérivées

Écrire une fonction `delta` qui étant donné une liste de `Integer` retourne une autre liste de `Integer` dont chaque élément est la différence de deux éléments consécutifs de la liste donnée. Vous pouvez supposer que la liste donnée a au moins deux éléments. (2 points)

Exemples d'utilisation:

```
*Main> delta [1,5,6,2,3]
[4,1,-4,1]
-- NB [5-1, 6-5, 2-6, 3-2]
*Main> delta [1..5]
[1,1,1,1]
*Main> delta [1,4..15]
[3,3,3,3]
```

### 4 Médian

Écrire une fonction qui étant donnée une liste retourne son élément médian, c'est-à-dire, un élément  $x$  tel que  $\leq 50\%$  des éléments de la liste sont supérieurs à  $x$  et  $\leq 50\%$  des éléments de la liste sont inférieurs à  $x$ . **NB:** si la liste a taille paire, il y a deux éléments médians possibles que vous pouvez retourner. Vous pouvez supposer que les éléments de la liste donnée sont distincts (pas de doublons). (2 points)

Exemples d'utilisation:

```
*Main> median [1..9]
5
*Main> median [9,8..1]
5
*Main> median [1,3,2,4,5,6,7]
4
```

### 5 Map, Fold, etc.

Quels sont les résultats d'évaluation des lignes suivantes ? (4 points)

```
filter (>5) [ x+y | x<-[1..3], y<-[2..6] ]
```

```
map (((+) . head) [2..5]) [3..6]
```

```
foldl (\x _ -> (head x)+(head (tail x)):x) [1,1] "Haskell!"
```

```
zipWith (^) [1..5] [2..4]
```

## 6 Polynômes

Pour cet exercice vous devez donner un programme de Haskell qui peut manipuler des polynômes. On utilise la déclaration suivante :

```
type Poly = [(Int, Int)]
```

L'idée est qu'on va représenter un polynôme par une liste de paires de `Int`. Chaque paire représente un monôme : le premier élément est le degré du monôme, et le deuxième le coefficient. Donc, la paire  $(5, 3)$  représente le monôme  $3x^5$ , alors que la liste  $[(0, 7), (1, 3), (3, 5)]$  représente le polynôme  $5x^3 + 3x + 7$ . **NB:** Pour cet exercice on ne considère que des entier non-négatifs pour les coefficients et les degrés.

Une liste de type  $[(Int, Int)]$  est une représentation valide d'un polynôme si elle satisfait les conditions suivantes : (i) tous les monômes ont un coefficient strictement positif et un degré non-négatif (ii) pour chaque  $d \geq 0$  il existe au plus un monôme de degré  $d$  dans la liste.

Donc, les listes  $[(1, 1), (3, 0), (4, 2)]$  et  $[(1, 1), (1, 3)]$  ne sont pas valides.

On vous demande de programmer (au moins) les fonctions suivantes.

1. `evalPoly` qui étant donné un polynôme et un `Int`  $x_0$ , retourne la valeur du polynôme évalué pour  $x = x_0$ .
2. `addPoly` qui étant donné deux polynômes retourne une représentation valide de leur somme (qui est aussi un polynôme).
3. `multPoly` qui retourne une représentation valide du produit de deux polynômes.
4. `poly2string` retourne une chaîne de caractères qui est un "bon affichage" du polynôme donné. Un bon affichage satisfait au moins les conditions suivantes : (i) les monômes de degré  $d \geq 2$  et coefficient  $c$  sont affichés comme  $cx^d$  (ii) les monômes de degré  $d = 1$  sont affichés comme  $cx$  (iii) les monômes de degré  $d = 0$  sont affichés comme  $c$  (iv) pour les monômes avec coefficient  $c = 1$  on n'affiche pas le coefficient. Donc, par exemple, on affiche  $x^4 + 2x^2 + 5x + 7$  plutôt que  $1x^4 + 2x^2 + 5x^1 + 7x^0$ . Pour simplifier la tâche, vous pouvez afficher les monômes dans n'importe quel ordre.

(6 points)

Exemples d'utilisation :

```
1 *Main> poly2string [(0,4), (1,2), (3,1), (5,7)]
2 "7x^5+x^3+2x+4"
3 *Main> poly2string $ addPoly [(0,4), (1,2), (5,7)] [(1,6), (3,8)]
4 "7x^5+8x^3+8x+4"
5 -- NB : (7x^5+2x+1) + (8x^3+6x)
6 *Main> poly2string $ multPoly [(0,1), (1,2), (2,1)] [(0,1), (1,1)]
7 "x^3+3x^2+3x+1"
8 -- NB : (x^2+2x+1) * (x+1)
9 *Main> evalPoly [(0,5), (1,3), (2,2)] 4
10 49
11 -- NB : 2x^2+3x+5 pour x=4
```

## Annexe

### Consignes

- Essayer de justifier vos réponses aussi bien que possible.
- Pour toutes les fonctions que vous définissez, vous devez préciser une signature de type.
- Pour les exercices de programmation, vous avez le droit de définir d'autres fonctions secondaires en plus des fonctions demandées (mais il faut aussi bien préciser leurs types).
- Vous pouvez utiliser toutes les fonctions de Haskell de la liste suivante. (Pour chaque fonction on vous rappelle aussi son type). Toute autre fonction que vous utilisez doit être implémentée.

### Fonctions de base

```
(:) :: a -> [a] -> [a]
head :: [a] -> a
tail :: [a] -> [a]
last :: [a] -> a
(++) :: [a] -> [a] -> [a]
(!!) :: [a] -> Int -> a
length :: [a] -> Int
take :: Int -> [a] -> [a]
drop :: Int -> [a] -> [a]
reverse :: [a] -> [a]
repeat :: a -> [a]
(.) :: (b -> c) -> (a -> b) -> a -> c
($) :: (a -> b) -> a -> b
map :: (a -> b) -> [a] -> [b]
zip :: [a] -> [b] -> [(a, b)]
filter :: (a -> Bool) -> [a] -> [a]
concatMap :: (a -> [b]) -> [a] -> [b]
foldl :: (b -> a -> b) -> b -> [a] -> b
foldr :: (a -> b -> b) -> b -> [a] -> b
dropWhile :: (a -> Bool) -> [a] -> [a]
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
fst :: (a, b) -> a
snd :: (a, b) -> b
(^) :: (Num a, Integral b) => a -> b -> a
(+) :: Num a => a -> a -> a
(-) :: Num a => a -> a -> a
(*) :: Num a => a -> a -> a
(==) :: Eq a => a -> a -> Bool
(/=) :: Eq a => a -> a -> Bool
(/) :: Fractional a => a -> a -> a
div :: Integral a => a -> a -> a
mod :: Integral a => a -> a -> a
max :: Ord a => a -> a -> a
min :: Ord a => a -> a -> a
(<) :: Ord a => a -> a -> Bool
(<=) :: Ord a => a -> a -> Bool
id :: a -> a
```