TD 4 : Listes

NB : quelques exercices de ce TD proviennent du site `https://wiki.haskell.org/H-99:_Ninety-Nine_Haskell_Problems` (qui est fortement recommandé)

1 Combinations

Donner une fonction `combinations` qui étant donné un entier `k` et une liste `xs` retourne toutes les sous–listes de `xs` de taille `k`.

Exemple :

```
*Main> combinations 3 "abcde"
["abc","abd","abe","acd","ace","ade","bcd","bce","bde","cde"]
```

Hint : vous pouvez utiliser la fonction `map` ou la compréhension en listes, ou `filter`, ou .... En plus, vous pouvez utiliser la fonction du TD1 qui produit tous les sous-ensembles d'une liste.

Solution :

Avec `map`

```
combinations :: Int -> [a] -> [[a]]
combinations 0 _ = [[]]
combinations _ [] = []
combinations n (x:xs) = (map (x:) (combinations (n-1) xs)) ++ (combinations n xs)
```

Avec <- et `subset` (du TD1)

```
combinations2 :: Int -> [a] -> [[a]]
combinations2 n xs = [ ys | ys<-subset xs, length ys==n ]
```

Avec `filter` et `subset`

```
combinations3 :: Int -> [a] -> [[a]]
combinations3 n xs = filter ((==n).length) (subset xs)
```

2 DropWhile

La fonction `dropWhile` a le type `dropWhile :: (a -> Bool) -> [a] -> [a]`. Cette fonction prend comme argument une condition booléenne et une liste et supprime le préfixe maximum de la liste qui ne contient que des éléments qui satisfont la condition.

Exemple :

```
*Main> dropWhile even [2,4,6,1,3,5,2,4]
[1,3,5,2,4]
```

Donner une implémentation récursive de cette fonction.

Solution :

```haskell
dropWhile' :: (a->Bool) -> [a] -> [a]
dropWhile' _ [] = []
dropWhile' f (x:xs)
  | f x = dropWhile' f xs
  | otherwise = x:xs
```

3 Compression de listes

Donner une fonction `compress` qui étant donné une liste retourne la même liste avec tous les doublons consécutifs remplacés par une seule copie.

Exemple :

```
*Main> compress [1,2,3,3,2,2,2,5,3,4]
[1,2,3,2,5,3,4]
*Main> compress "aabaabaabbba"
"abababa"
```

1. Donner une implémentation récursive directe.
2. Donner une implémentation qui utilise `dropWhile`.
3. Donner une implémentation qui utilise `filter` et supprime tous les doublons (même non–consécutifs)
   Exemple :

   ```
   *Main> compress' [1,2,3,1,2,3,1,1,2,2,2,3,3,3,1,2]
   [1,2,3]
   ```

4. Donner une implémentation qui utilise `foldr`.
5. Donner une implémentation qui utilise `foldl`.

Solution :

1. ```haskell
   compress :: Eq a => [a] -> [a]
   compress [] = []
   compress [x] = [x]
   compress (x:y:ys)
    | x == y = compress (x:ys)
    | otherwise = x:compress (y:ys)
   ```

2. ```haskell
   compress'' :: Eq a => [a] -> [a]
   compress'' [] = []
   compress'' (x:xs) = x:(compress'' $ dropWhile (== x) xs)
   ```

3. ```haskell
   compress' :: Eq a => [a] -> [a]
   compress' [] = []
   compress' (x:xs) = x: (compress' $ filter (/= x) xs )
   ```

4. ```haskell
   compress''' :: Eq a => [a] -> [a]
   compress''' x = foldr (\a b -> if a == (head b) then b else a:b) [last x] x

   -- more verbose version
   maybeAdd :: Eq a => a -> [a] -> [a]
   maybeAdd x [] = [x]
   maybeAdd x (y:ys)
     | x==y = y:ys
   ```

```
    | otherwise = x:y:ys

compress'''' :: Eq a => [a] -> [a]
compress'''' xs = foldr maybeAdd [] xs
```

5. ```
   maybeAppend :: Eq a => [a] -> a -> [a]
   maybeAppend [] x = [x]
   maybeAppend xs x
     | x==(last xs) = xs
     | otherwise = xs ++ [x]

   compress5 :: Eq a => [a] -> [a]
   compress5 xs = foldl maybeAppend [] xs
   ```

4 concatMap

La fonction `concatMap` a comme type `concatMap :: (a -> [b]) -> [a] -> [b]`. Étant donné une liste d'éléments de type `a`, et une fonction qui s'applique sur de tels éléments et retourne une liste de `b`, applique cette fonction sur chaque élément et retourne la concaténation de tous les résultats.

Exemple :

```
*Main> concatMap show [12,34,56]
"123456"
```

Donner une implémentation récursive de cette fonction

Solution :

```
concatMap' :: (a->[b]) -> [a] -> [b]
concatMap' _ [] = []
concatMap' f (x:xs) = (f x) ++ (concatMap f xs)
```

5 Réplication

Écrire une fonction `replic` qui étant donné une liste `xs` et un entier `n` retourne la même liste où chaque élément est répété `n` fois.

Exemple :

```
*Main> replic  [1..5] 5
[1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5]
```

1. Donner une implémentation récursive directe. Vous pouvez utiliser les fonction `take` et `repeat`.
2. Donner une implémentation qui utilise `concatMap`.
3. Donner une implémentation qui utilise `foldr`.

Solution :

1. ```
   replic :: [a] -> Int -> [a]
   replic [] _ = []
   replic (x:xs) n = (take n (repeat x)) ++ (replic xs n)
   ```

2. ```
   replic2 :: [a] -> Int -> [a]
   replic2 xs n = concatMap (take n . repeat) xs
   ```
   ```

3.
```
replic3 :: [a] -> Int -> [a]
--replic3 xs n = foldr (\x ys-> take n (repeat x) ++ ys) [] xs
replic3 xs n = foldr  ((++).(take n).(repeat))  [] xs
```