TD de programmation fonctionnelle et logique
Corrigé du TD 5 : fonctionnelles sur listes

Polynômes creux

Dans ce TD nous allons manipuler des monômes définis par le type suivant :

```
# type monôme = {coefficient : int; degré : int};;
type monôme =  coefficient : int; degré : int;
```

Un polynôme (creux) sera alors constitué d'une liste de monômes, cette liste n'étant pas supposée être triée en fonction des degrés des monômes.

Affichage

1. Écrivez une fonction d'affichage d'un monôme.

```
# let affiche_monôme m =
    print_string
      ((string_of_int m.coefficient)^"."X^"^(string_of_int m.degré));;
val affiche_monôme : monôme -> unit = <fun>
```

2. Écrivez, au moyen d'une fonctionnelle, une fonction qui affiche tous les monômes d'un polynôme.

```
# let affiche_polynôme p =
     let affiche_aux m =
        affiche_monôme m; print_string " "
     in List.iter affiche_aux p;;
val affiche_polynôme : monôme list -> unit = <fun>
```

Produit de deux polynômes

1. Écrivez une fonction de multiplication de deux monômes.

```
# let mul_monôme m n =
    {coefficient = m.coefficient*n.coefficient; degré = m.degré+n.degré};;
val mul_monôme : monôme -> monôme -> monôme = <fun>
```

2. Écrivez, au moyen d'une fonctionnelle, une fonction qui multiplie un polynôme par un monôme.

```
# let mul_polynôme_par_monôme m p =
    List.map (mul_monôme m) p;;
val mul_polynôme_par_monôme : monôme -> monôme list -> monôme list = <fun>
```

3. Écrivez, au moyen d'une fonctionnelle, une fonction qui multiplie deux polynômes.

```
# let mul_polynôme p q =
    let l = List.map (function x -> mul_polynôme_par_monôme x p) q
    in List.fold_left (function x -> function y -> x @ y) [] l;;
val mul_polynôme : monôme list -> monôme list -> monôme list = <fun>
```

```
# let mul_polynôme p q =
    List.fold_left
      (function x -> function y -> x @ y)
```

```
      []
      (List.map (function x -> mul_polynôme_par_monôme x p) q);;
val mul_polynôme : monôme list -> monôme list -> monôme list = <fun>
```

4. Même question que précédemment, mais en n'utilisant qu'une seule fonctionnelle... (si vous en avez utilisé deux à la question précédente).

```
# let mul_polynôme p q =
    List.fold_left
      (function x -> function y -> (mul_polynôme_par_monôme y p) @ x)
      []
      q;;
val mul_polynôme : monôme list -> monôme list -> monôme list = <fun>

# let mul_polynôme p q =
    List.fold_right
      (function x -> function y -> (mul_polynôme_par_monôme x p) @ y)
      q
      [];;
val mul_polynôme : monôme list -> monôme list -> monôme list = <fun>
```

Évaluation d'un polynôme en un point

1. Écrivez une fonction qui évalue un monôme en un point.

```
# let eval_monôme m x =
    let rec puiss = function
        0 -> 1
      | n -> x * (puiss (n-1))
    in m.coefficient * (puiss m.degré);;
val eval_monôme : monôme -> int -> int = <fun>
```

2. Écrivez, au moyen d'une fonctionnelle, une fonction qui évalue un polynôme en un point.

```
# let eval_polynôme p x =
    List.fold_right
      (function y -> function z -> z + (eval_monôme y x))
      p
      0;;
val eval_polynôme : monôme list -> int -> int = <fun>

# let eval_polynôme p x =
    List.fold_left
      (function y -> function z -> y + (eval_monôme z x))
      0
      p;;
val eval_polynôme : monôme list -> int -> int = <fun>
```