TD de programmation fonctionnelle et logique
Corrigé du TD 2 : récursivité

11 décembre 2002

Expressions à évaluer

```
# let x = 1;;

# let y = 4;;

# let x = 3 and y = x + 1 in
    let x = x + y and y = x - y
      in (x,y);;

# let x = 3
    in let  y = x + 1
      in let x = x + y
        in let y = x - y
          in (x,y);;
```

```
# let x = 1;;
val x : int = 1

# let y = 4;;
val y : int = 4

# let x = 3 and y = x + 1 in
    let x = x + y and y = x - y
      in (x,y);;
- : int * int = 5, 1

# let x = 3
    in let  y = x + 1
      in let x = x + y
        in let y = x - y
          in (x,y);;
- : int * int = 7, 3
```

Fonctions élémentaires sur les listes

Écrire une fonction :

1. Qui rend `true` si et seulement si une liste n'a qu'un seul élément ;

```
# let un_seul_élément = function
     []      -> false
   | t::r::l -> false
   | _       -> true;;
val un_seul_élément : 'a list -> bool = <fun>

# let un_seul_élément = function
     [a] -> true
   |  _  -> true;;
val un_seul_élément : 'a list -> bool = <fun>
```

2. Qui renvoie le second élément d'une liste d'au moins deux éléments ;

```
# let second_élément = function
     []      -> failwith "second_élément"
   | t::r::l -> r
   | _       -> failwith "second_élément";;
val second_élément : 'a list -> 'a = <fun>

# let second_élément = function
```

```
  | t::r::l -> r
  | _       -> failwith "second_élément";;
val second_élément : 'a list -> 'a = <fun>
```

3. Qui calcule la longueur d'une liste ;

```
# let rec longueur = function
    []   -> 0
  | t::r -> 1+ longueur r;;
val longueur : 'a list -> int = <fun>
```

4. Qui calcule la somme des éléments d'une liste de nombres ;

```
# let rec somme = function
    []   -> 0
  | t::r -> t + somme r;;
val somme : int list -> int = <fun>
```

5. Qui rend le dernier élément d'une liste non vide ;

```
# let rec dernier = function
    []    -> failwith "dernier"
  | t::[] -> t
  | _::r  -> dernier r;;
val dernier : 'a list -> 'a = <fun>
```

6. Qui prend une liste et un élément en arguments et renvoie `true` si et seulement si l'élément appartient à la liste ;

```
# let rec appartient_a x = function
    []   -> false
  | t::r -> t=x or appartient_a x r;;
val appartient_a : 'a -> 'a list -> bool = <fun>
```

7. Qui teste l'égalité de deux listes ;

```
# let rec égales l1 l2 = match (l1,l2) with
        [],[]     -> true
  | t1::r1,t2::r2 -> t1=t2 & égales r1 r2
  |      _,_      -> false;;
val égales : 'a list -> 'a list -> bool = <fun>
```

```
# let rec égales l1 l2 = match (l1,l2) with
        [],[]     -> true
  | t1::r1,t2::r2 -> t1=t2 & égales r1 r2
  |       _      -> false;;
val égales : 'a list -> 'a list -> bool = <fun>
```

8. Qui concatène deux listes ;

```
# let rec concatene l1 l2 =
  match l1 with
    []   -> l2
  | t::r -> t :: concatene r l2 ;;
val concatene : 'a list -> 'a list -> 'a list = <fun>
```

9. Qui renverse une liste (écrire une fonction avec et une fonction sans accumulateur) ;

```
# let rec renverse = function
    []   -> []
  | t::r -> concatene (renverse r)  [t] ;;
val renverse : 'a list -> 'a list = <fun>
# let renverse l =
  let rec renverse_accu accu = function
      []   -> accu
    | t::r -> renverse_accu (t::accu) r
```

```
   in renverse_accu [] l;;
val renverse : 'a list -> 'a list = <fun>
```

10. Qui insère un élément dans une liste triée (la liste obtenue doit bien évidemment être triée) ;

```
# let rec insere x = function
     []       -> [x]
  | t::r as l -> if x<t then x::l
                        else t:: insere x r;;
val insere : 'a -> 'a list -> 'a list = <fun>
```

11. Qui trie par insertion une liste.

```
# let rec tri = function
     []  -> []
  | t::r -> insere t (tri r);;
val tri : 'a list -> 'a list = <fun>

# tri [6;2;8;4;9];;
- : int list = [2; 4; 6; 8; 9]
```