TD 3

Liste et récursivité

Correction

Exercice 1 :

1. Écrire une fonction `der` qui renvoie le dernier terme d'une série.

   `val der : 'a list -> 'a = <fun>`

Correction

Listing 1 – La fonction `der`

```
let rec der liste =match liste with
  |[] -> failwith "liste␣vide␣dans␣der"
  |[a] -> a
  |a::reste -> der reste ;;
```

2. Écrire une fonction `avant_der` qui renvoie l'avant dernier terme d'une série.

   `val avant_der : 'a list -> 'a = <fun>`

Correction

Listing 2 – La fonction `avant_der`

```
let rec avant_der liste =match liste with
  |[] | [_]-> failwith "erreu␣liste␣avant_der"
  |[a ; b] ->  a
  |a::reste -> avant_der reste ;;
```

3. Écrire une fonction `nieme` qui renvoie le nième terme d'une série pour un n donné.

   `val nieme : 'a list -> int -> 'a = <fun>`

Correction

Listing 3 – La fonction `nieme`

```
let rec nieme liste n =
    if (liste = [] || (n <= 0 ))
      then failwith "␣erreurs␣"
    else
      begin
        if n = 1 then List.hd liste
        else nieme (List.tl liste) (n-1)
      end ;;

(* version pattern matching *)
let rec nieme liste n = match liste with
  | [] -> failwith "␣erreurs␣"
  | a::reste when n <= 0 -> failwith "␣erreurs␣"
  | a::reste when n = 1 -> a
  | a::reste  -> nieme reste (n-1);;
```

Exercice 2 : Écrire la fonction `minimax` qui donne le plus grand et le plus petit élément d'une liste :

```
val minimax : 'a list -> 'a * 'a = <fun>
```

Correction

Listing 4 – La fonction `minimax`

```
let rec minimax liste = match liste with
  | [] -> failwith "␣liste␣vide␣"
  | [a] -> (a , a)
  |a::reste ->  let ( min_reste , max_reste ) = minimax reste in
      if a > max_reste then ( min_reste , a ) else
      if a < min_reste then ( a , max_reste ) else
      ( min_reste , max_reste ) ;;
```

Exercice 3 : Écrire la fonction `cat` qui concatène deux listes.

```
val cat : 'a list -> 'a list -> 'a list = <fun>
```

Correction

Listing 5 – La fonction `cat`

```
let rec cat liste_1 liste_2 =match liste_1 with
  | [] -> liste_2
  | a::reste -> a::( cat reste liste_2 );;
```

Exercice 4 : Écrire la fonction `map_liste` qui applique une fonction à tous les éléments d'une liste, et renvoie la liste modifiée.

```
val map_liste : ('a -> 'b) -> 'a list -> 'b list = <fun>
map_liste (fun x -> ( x > 5) ) [7;4;3;1;9;7;4;6];;
- : bool list = [true; false; false; false; true; true; false; true]
```

Correction

Listing 6 – La fonction `map_tab`

```
let rec map_liste f liste = match liste with
  | [] -> []
  | a::reste -> ( f a ) :: (map_liste f reste );;
```

Exercice 5 : Écrire une fonction `occurrence` tel que `occurrence element liste` retourne la liste des positions des occurrences de élément dans liste.

```
val occurrence : 'a -> 'a list -> int list = <fun>
occurrence "a" [ "a";"bb";"a";"jj";"n";"a"];;
- : int list = [1; 3; 6]
```

Correction

Listing 7 – La fonction occurrence

```
let occurrence element liste =
  let rec aux liste position = match liste with
    | [] -> []
    | a::reste when a = element ->
      position :: (aux reste (position +1 ))
    | a::reste -> aux reste (position +1 )
  in
  aux liste 1;;
```

Exercice 6 : Soit $n \in \mathbb{N}^*$ et $l = (x_i)_{1\leq i\leq n}$ une liste de n éléments.
Un palier de l est une liste d'éléments consécutifs égaux , et de longueur maximale pour cette propriété.
Ainsi, la liste $[2;2;8;3;3;3]$ comporte trois paliers : $[2;2]$, $[8]$, $[3;3;3]$.
On peut donner une définition par induction structurelle :

- La liste vide ne comporte aucun palier.
- Soit $l = (x_i)_{1\leq i\leq n}$ une liste non vide ; notons j le plus grand indice tel que $x_i = x_1$ pour tout $i \in [\![1;j]\!]$: la liste $(x_i)_{1\leq i\leq j}$ est le premier palier de l ; si $j = n$, c'est le seul ; sinon les autres paliers sont ceux de $(x_i)_{j<i\leq n}$.

1. Écrire une fonction `palier_tete` qui renvoie le couple formé du premier palier et la reste de la liste.

   ```
   val palier_tete : 'a list -> 'a list * 'a list = <fun>
   ```

   Correction

Listing 8 – La fonction palier_tete

```
let rec palier_tete liste = match liste with
  | [] -> ([],[])
  | [a] -> ( [a] , [])
  | a::b::reste when  a=b   ->
      let (tete,r) = palier_tete (b::reste) in
      (a::tete, r )
  | a::b::reste -> ([a] , b::reste );;
```

2. Écrire une fonction `palier` qui renvoie la liste des paliers de la liste entrée.

   ```
   val palier : 'a list -> 'a list list = <fun>
   ```

   Correction

Listing 9 – La fonction palier

```
let rec palier liste = if liste = [] then []
  else
    begin
      let (pal_tete, reste ) = palier_tete liste in
      let l_reste = palier reste in
      pal_tete :: l_reste
    end;;
```

Exercice 7 : **Tri par sélection :**

Le tri par sélection (ou tri par extraction) est un des algorithmes de tri les plus triviaux. Il consiste en la recherche soit du plus grand élément (ou le plus petit) que l'on va replacer à sa position finale c'est-à-dire en dernière position (ou en première), puis on recherche le second plus grand élément (ou le second plus petit) que l'on va replacer également à sa position finale c'est-à-dire en avant-dernière position (ou en seconde), etc., jusqu'à ce que le tableau, ou la liste, soit entièrement trié.

1. Écrire la fonction `min_list` qui renvoie pour une liste donnée l le minimum de la liste ainsi que le reste de la liste.

   ```
   val min_list : 'a list -> 'a * 'a list = <fun>
   ```

   Correction

Listing 10 – La fonction `min_list`

```
let rec min_list liste = match liste with
  | [] -> failwith ("pas d'element ")
  | [a] -> (a,[])
  | a::reste ->
    let (min_reste,reste_reste ) = min_list reste in
      if a < min_reste then (a, reste )
      else (min_reste,a::reste_reste );;
```

2. Écrire la fonction `tri_selec` qui renvoie la liste triée par sélection :

   ```
   val tri_selec : 'a list -> 'a list = <fun>
   ```

   Correction

Listing 11 – La fonction `tri_selec`

```
let rec tri_select liste = match liste with
  | [] -> []
  | l -> let (m,r) = min_list l in
      m :: tri_select  r;;
tri_select [ 9;4;2;1;3;4;6;7];;
```