TD 6 Programmation fonctionnelle
Mutation et style impératif

1 Tableaux

1. Écrivez une fonction moyenne : float array -> float qui renvoie la moyenne d'un tableau de flottants. La fonction renvoie une erreur avec invalid_arg si le tableau est de longueur 0.

Solution.

```
let moyenne t =
  let n = Array.length t in
    if n = 0
    then  invalid_arg "moyenne"
    else
      let sum = ref 0. in
        for i = 0 to n-1 do
          sum := !sum +. t.(i)
        done;
        !sum /. float(n)
```

□

2. Écrivez une fonction scalaire : float array -> float array -> float qui calcule le produit scalaire de deux tableaux de même longueur. On rappelle que le produit scalaire de $(a_1, \ldots, a_n)$ et $(b_1, \ldots, b_n)$ vaut $\Sigma_{i=1}^{n} a_i b_i$.

Solution.

```
let scalaire t1 t2 =
  let res = ref 0. in
    for i=0 to (Array.length t1) - 1 do
      res := !res +. t1.(i) *. t2.(i)
    done;
    !res
```

□

2 Tri sélection

1. Définissez la fonction swap : 'a array -> int -> int -> unit telle que swap t i j échange les valeurs d'indice i et j de t.

Solution.

```
let swap t i j =
  let tmp = t.(i) in
    t.(i) <- t.(j);
    t.(j) <- tmp
```

□

2. Définissez la fonction argmin : int −> 'a array −> int telle que argmin k [|a0 ;...; an|] renvoie l'indice du plus petit ai pour i compris entre k et n (les k premiers éléments ne sont pas pris en compte). Par exemple, argmin 2 [|0;2;5;1;4|] renvoie 3.

Solution.

```
let argmin k t =
  let res = ref k in
    for i=k+1 to Array.length t - 1 do
      if t.(i) < t.(!res)
      then res := i
    done;
    !res
```

□

3. Déduisez-en une fonction tri_selection : 'a array −> unit qui trie un tableau en appliquant le tri sélection.

Solution.

```
let tri_selection t =
  for i = 0 to Array.length t - 2 do
    swap t i (argmin i t)
  done
```

□

3 Style impératif et style itératif

Programmez en style impératif, sans utiliser la récurrence, uniquement à l'aide de boucles, les fonctions suivantes

1. la fonction factorielle : int −> int qui calcule la factorielle d'un nombre

Solution.

```
let factorielle n =
  let res = ref 1 in
    for i = 2 to n do
      res := !res * i
    done;
    !res
```

□

2. la fonction somme: int list −> int qui calcule la somme d'une liste ;

Solution.

```
let somme l =
  let res = ref 0 in
  let li = ref l in
    while !li <> [] do
      res := !res + List.hd !li;
      li := List.tl !li
```

```
    done;
    !res
```

□

3. la fonction argmin : 'a list −> int qui calcule l'indice du minimum d'une liste non vide. En cas de liste vide, une erreur est levée avec invalid_arg .

Solution.

```
let argmin l =
  if l = [] then invalid_arg "argmin"
  else begin
    let res = ref 0 in
    let min = ref (List.hd l) in
    let li = ref l in
    let i = ref 0 in
      while !li <> [] do
        if List.hd !li < !min then begin
          min := List.hd !li;
          res := !i
        end;
        i := !i + 1;
        li := List.tl !li
      done;
      !res
  end
```

□

4. Reprogrammez ces fonctions sans utiliser de référence et avec des fonctions récursives.

Solution.

```
let factorielle n =
  let rec fact res i =
    if i = n+1 then res
    else fact (res*i) (i+1)
  in fact 1 1

let somme l =
  let rec som res li = match li with
    | [] -> res
    | x::ll -> som (res+x) ll
  in som 0 l

let argmin l =
  let rec am res min li i =
    match li with
      | [] -> res
      | x::ll -> if x<min then am i x ll (i+1)
          else am res min ll (i+1)
  in match l with
    | [] -> invalid_arg "argmin"
    | x::ll -> am 0 x ll 1
```

□

4 Listes mutables cycliques

On considère le type suivant

```
type mlist =
  | Empty
  | Cons of int ref * mlist ref
```

1. Définissez la fonction somme : mlist -> int

   Solution.

   ```
   let rec somme ml = match ml with
     | Empty -> 0
     | Cons(i,ml2) -> !i + somme !ml2
   ```

   □

2. Définissez la fonction set : mlist -> int -> int -> unit telle que set l i n remplace l'entier à l'indice i par n, et fait une erreur avec invalid_arg si l'indice n'est pas correct.

   Solution.

   ```
   let rec set ml i n = match ml with
     | Empty -> invalid_arg "set"
     | Cons(r, ml2) -> if i=0 then r:=n else set !ml2 (i-1) n
   ```

   □

3. Définissez la fonction cycle : mlist -> unit qui transforme une liste acyclique non vide en une liste cyclique.

   Solution.

   ```
   let cycle ml =
     let rec cyc = function
       | Empty -> invalid_arg "cycle"
       | Cons(_,r) -> if !r=Empty then r:=ml else cyc !r
     in cyc ml
   ```

   □

4. Définissez la fonction clength : mlist -> unit qui renvoie la longueur d'une liste cyclique ou acyclique.

   Solution.

   ```
   let clength ml =
     let rec clen i = function
       | Empty -> i
       | Cons(_,mll) -> if ml == !mll then i+1 else clen (i+1) !mll
     in clen 0 ml
   ```

   □