Corrigé du DS2

Exercice 1 : Une preuve de Cauchy de l'inégalité entre moyennes arithmétique et géométrique

1 1 est l'unique cas de base, et les constructeurs sont $n \mapsto 2n$ et $n \mapsto n-1$ (envoyant 1 sur 1 par exemple, pour que la fonction soit définie sur $\mathbb{N}^*$).

```
# let rec cauchy = function
        | 1 -> true
        | n -> if n mod 2 = 0 then cauchy (n / 2) else cauchy (n + 1);;
cauchy : int -> bool = <fun>

# trace "cauchy";;
The function cauchy is now traced.
- : unit = ()

# cauchy 45;;
cauchy <-- 45
cauchy <-- 46
cauchy <-- 23
cauchy <-- 24
cauchy <-- 12
cauchy <-- 6
cauchy <-- 3
cauchy <-- 4
cauchy <-- 2
cauchy <-- 1
cauchy --> true
cauchy --> true
cauchy --> true
cauchy --> true
cauchy --> true
cauchy --> true
cauchy --> true
cauchy --> true
cauchy --> true
cauchy --> true
cauchy --> true
- : bool = true
```

Exercice 2 : Programmation sur les listes

```
(* 2 *)

# let rec conc l l' = match l with
        | [] -> l'
        | a :: q -> a :: conc q l';;
```

```
conc : 'a list -> 'a list -> 'a list = <fun>

# conc [1; 2; 3] [4; 5; 6];;
- : int list = [1; 2; 3; 4; 5; 6]

(* 3 *)

# let rec membre elt = function
        | [] -> false
        | a :: q -> elt = a or membre elt q;;
membre : 'a -> 'a list -> bool = <fun>

# membre 3 [1; 2; 3; 4];;
- : bool = true

# membre 5 [1; 2; 3; 4];;
- : bool = false

(* 4 *)

# let rec tous_distincts = function
        | [] -> true
        | a :: q when (membre a q) -> false
        | _ :: q -> tous_distincts q;;
tous_distincts : 'a list -> bool = <fun>

# tous_distincts [1; 2; 5; 3; 4; 5];;
- : bool = false

(* 5 *)

# let tourne = function
        | [] -> []
        | a :: q -> conc q [a];;
tourne : 'a list -> 'a list = <fun>

(* 6.a *)

# let rec place_en_tete elt = function
        | l when not membre elt l -> failwith "impossible"
        | l when hd l = elt -> l
        | l -> place_en_tete elt (tourne l);;
place_en_tete : 'a -> 'a list -> 'a list = <fun>

# let test = [1; 2; 3; 4; 5; 6];;
test : int list = [1; 2; 3; 4; 5; 6]

# let test' = place_en_tete 3 test;;
test' : int list = [3; 4; 5; 6; 1; 2]

(* 6.b *)

# let permutee_circulaire l = if not tous_distincts l
        then failwith "argument incorrect"
        else
function
        | [] -> l = []
        | l' when not (tous_distincts l') -> failwith "argument incorrect"
        | a' :: q' as l' -> (membre a' l) && (place_en_tete a' l = l');;
```

```
permutee_circulaire : 'a list -> 'a list -> bool = <fun>

# permutee_circulaire test test';;
- : bool = true

(* 6.c *)

# let rec permutee l = function
        | [] -> l = []
        | a' :: q' as l' -> (membre a' l) && (permutee q' (tl (place_en_tete a' l)));;
permutee : 'a list -> 'a list -> bool = <fun>

# permutee [1; 2; 3; 4; 5] [1; 3; 2; 5; 4];;
- : bool = true
```

Exercice 3 : Parcours en escalier dans un tableau

```
(* 1 *)

# let init n = let temp = make_matrix (n + 1) (n + 1) 1 in
        for i = 1 to n do temp.(i).(0) <- 2 * temp.(i - 1).(0) done;
        for j = 1 to n do temp.(0).(j) <- 3 * temp.(0).(j - 1) done;
        for i = 1 to n do for j = 1 to n do
                temp.(i).(j) <- temp.(i).(0) * temp.(0).(j)
        done done;
        temp;;
init : int -> int vect vect = <fun>

(* 2 *)

# let plus_grand_minorant n p = let M = init n in

        let j = ref n in while M.(0).(!j) > p do j := !j - 1 done;

        let result = ref M.(0).(!j) and temp = ref M.(0).(!j) and i = ref 0 in
        while !temp < p & !i < n do
                i := !i + 1;
                let k = M.(!i).(!j) in
                        if k <= p or !j = 0
                        then (temp := k; if k <= p && !result <= k then result := k;)
                        else
                        (j := !j - 1; temp := M.(!i).(!j);)
                done;
        !result;;
plus_grand_minorant : int -> int -> int = <fun>
```