Correction exam programmation fonctionnelle

```
type date = {day:int; month:int; year:int}

type time = {mutable hours:int; mutable minutes:int}

type id = {name:string; surname:string; age:int}

type arbre =
| Const of int
| Var of string
| Plus of arbre * arbre

(*****************************************************************************************)

let milieu (x1,y1) (x2,y2) =
  ((x1 +. x2) /. 2., (y1 +. y2) /. 2.)


let distance (x1,y1) (x2,y2) =
  sqrt ((x2 -. x1) ** 2. +. (y2 -. y1) ** 2.)


let somme (x1,y1) (x2,y2) =
  (x1 +. x2),(y1 +. y2)


let date_to_string date =
  Format.sprintf "%d/%d/%d" date.day date.month date.year


let incr_time time =
  if time.minutes=59
  then begin
    time.minutes <- 0;
    time.hours <- time.hours + 1
  end
  else time.minutes <- time.minutes + 1


let reset_time time = {time with minutes=0}


let rev_array t =
  let n = Array.length t in
  Array.init n (fun i -> t.(n-i-1))
```

```
let somme_array t =
  let n = Array.length t in
  if n = 0 then [||] else begin
    let res = Array.make n t.(0) in
    for i=1 to n-1 do
      res.(i) <- res.(i-1) + t.(i)
    done;
    res
  end


let un_sur_deux t =
  let n = Array.length t in
  Array.init ((n+1)/2) (fun i-> t.(2*i))


let rec paire_list = function
| [] -> []
| [x] -> [(x,x)]
| x::y::tl -> (x,y)::paire_list tl


let rec moyenne2_list = function
| [] -> []
| [x] -> [x]
| x::y::tl -> (x +. y) /. 2. :: moyenne2_list tl


let rec swap2_list = function
| [] -> []
| [x] -> [x]
| x::y::tl -> y::x::swap2_list tl


let rec temps_de_vol n =
   if n=1 then 1
   else if n mod 2 = 0 then 1+temps_de_vol (n/2)
   else 1+temps_de_vol (3*n+1)


let rec altitude_max n =
  if n=1 then 1
  else if n mod 2 = 0 then max n (altitude_max (n/2))
  else max n (altitude_max (3*n+1))


let rec nombre_descentes n =
```

```
  if n=1 then 0
  else if n mod 2 = 0 then 1+nombre_descentes (n/2)
  else nombre_descentes (3*n+1)


let plus_longue_sous_suite_croissante l =
  let rec iter res cur l =
    match l with
    | [] -> res
    | [x] -> max res cur
    | x::y::tl when x<=y -> iter res (cur+1) (y::tl)
    | x::y::tl -> iter (max res cur) 1 (y::tl)
  in iter 0 1 l


let plus_longue_sous_suite_paire l =
  let rec iter res cur l =
    match l with
    | [] -> max res cur
    | hd::tl when hd mod 2 = 0 -> iter res (cur+1) tl
    | hd::tl -> iter (max res cur) 0 tl
  in iter 0 0 l


let plus_longue_sous_suite_constante l =
  let rec iter res cur l =
    match l with
    | [] -> res
    | [x] -> max res cur
    | x::y::tl when x=y -> iter res (cur+1) (y::tl)
    | x::y::tl -> iter (max res cur) 1 (y::tl)
  in iter 0 1 l


let sort_by_name l =
  let comp id1 id2 = compare id1.name id2.name in
  List.sort comp l


let sort_by_surname l =
  let comp id1 id2 = compare id1.surname id2.surname in
  List.sort comp l


let sort_by_age l =
  let comp id1 id2 = compare id1.age id2.age in
  List.sort comp l
```

```
let rec nb_feuilles = function
| Const _ | Var _ -> 1
| Plus(a1,a2) -> nb_feuilles a1 + nb_feuilles a2


let rec hauteur = function
| Const _ | Var _ -> 0
| Plus(a1,a2) -> 1+max (hauteur a1) (hauteur a2)


let rec liste_var = function
| Const _ -> []
| Var x -> [x]
| Plus(a1,a2) -> liste_var a1 @ liste_var a2


let is_int s =
  try ignore (int_of_string s);true
  with Failure _ -> false


let last_line ic =
  let res = ref "" in
  let rec iter () =
    try
      res := input_line ic;
      iter ()
    with
    | End_of_file -> close_in ic; !res
  in iter ()


let rec nth_line ic n =
  try
    if n=1 then input_line ic
    else begin
      ignore (input_line ic);
      nth_line ic (n-1)
    end
  with
  | End_of_file -> ""


let dominos l =
  let rec iter goal passed todo = match todo with
  | [] -> passed = []
  | (x,y)::tl when x=goal ->
```

```
  iter y [] (passed@tl) || iter goal ((x,y)::passed) tl
| (x,y)::tl when y=goal ->
  iter x [] (passed@tl) || iter goal ((x,y)::passed) tl
| (x,y)::tl -> iter goal ((x,y)::passed) tl in
let rec f l1 l2 = match l1 with
| [] -> l2=[]
| (x,y)::tl -> iter x [] (tl@l2) || iter y [] (tl@l2) || f tl ((x,y)::l2) in
f l []
```