Correction du DS 2

Julien Reichert

La correction des questions de cours étant dans le cours, elle ne sera pas donnée ici. De même, les exercices issus des TD et TP ont leur correction déjà publiée.

Exercices

Exercice 1 :

```
exception Perdu

let alternance tab =
  let n = Array.length tab in n < 2 || tab.(0) <> tab.(1)
  && let variation = tab.(1) > tab.(0) in
  try
    for i = 1 to n - 2 do
      if tab.(i+1) = tab.(i) || ((tab.(i+1) > tab.(i)) = variation) <> (i mod 2 = 0) then raise Perdu
    done; true
  with Perdu -> false
```

Exercice 2 :

```
let alternance2 l =
  let rec alternance2aux attendu l =
    match l with
    | [] | [ _ ] -> true
    | a::b::q -> a <> b && (a > b) = attendu && alternance2aux (not attendu) (b::q)
  in match l with
  | [] | [ _ ] -> true
  | a::b::q -> a <> b && alternance2aux (a < b) (b::q)
```

Exercice 3 :

```
let decoupage ch = let chaine = ch ^ " " in (* astuce pour gérer le dernier mot *)
  let nbm = ref 0 in let pltm = ref 0 in let idplm = ref (-1) in let iplm = ref (-1) in let tm = ref 0 in
  for i = 0 to String.length chaine - 1 do
    if chaine.[i] = ' ' then
    begin
      if !tm > !pltm then
      begin
        pltm := !tm; iplm := !nbm; idplm := i - !tm
      end;
      tm := 0; incr nbm;
    end
    else incr tm
  done;
  (!nbm, !pltm, !idplm, !iplm)
```

Exercice 4 :

```
struct effectifsommesommecarres { int effectif ; int somme ; int sommecarres ; };
typedef struct effectifsommesommecarres essc;

essc effectif_somme_sommecarres(int* tab, int taille)
{
  essc rep = { .effectif = taille , .somme = 0 , .sommecarres = 0 };
  for (int i = 0 ; i < taille ; i += 1)
  {
    rep.somme += tab[i];
    rep.sommecarres += tab[i] * tab[i];
  }
  return rep;
}
```

Problème

Question P1 :

```
let exemple = Array.make_matrix 9 9 (0, false)
let _ =
  exemple.(0).(4) <- (1, true); exemple.(8).(4) <- (1, true);
  exemple.(4).(0) <- (1, true); exemple.(4).(8) <- (1, true);
  exemple.(1).(4) <- (2, false); exemple.(7).(4) <- (2, false);
  exemple.(4).(1) <- (2, false); exemple.(4).(7) <- (2, false);
  exemple.(2).(4) <- (1, false); exemple.(6).(4) <- (1, false);
  exemple.(4).(2) <- (1, false); exemple.(4).(6) <- (1, false);
  exemple.(3).(4) <- (1, false); exemple.(5).(4) <- (1, false);
  exemple.(4).(3) <- (1, false); exemple.(4).(5) <- (1, false);
  exemple.(4).(4) <- (3, false)
let instance_exemple = exemple, (ref 4, ref 4, ref 3)
```

Question P2 :

```
let correcte (grille, (l, c, _)) =
  fst grille.(!l).(!c) <> 0
  && Array.for_all (Array.for_all (fun (c, b) -> c <> 0 || not b)) grille
```

Question P3 :

```
let reduite (grille, _) =
  let n = Array.length grille in let m = Array.length grille.(0) in
  let pl = ref false in
  let dl = ref false in
  let pc = ref false in
  let dc = ref false in
  for i = 0 to m-1 do
    if fst grille.(0).(i) <> 0 then pl := true;
    if fst grille.(n-1).(i) <> 0 then dl := true
  done;
  for i = 0 to n-1 do
    if fst grille.(i).(0) <> 0 then pc := true;
    if fst grille.(i).(m-1) <> 0 then dc := true
  done;
  !pl && !dl && !pc && !dc
```

Question P4 :

```
let solution_exemple =
[|
 [| (-2, 3) ; (-1, 0) ; (1, 2) ; (0, 0) |];
 [| (-1, 0) ; (-3, 0) ; (-3, 0) ; (-1, 0) |]
|]
```

Question P5 :

```
let fini grille =
  Array.for_all (Array.for_all (fun (c, b) -> not b)) grille
```

Question P6 :

```
let gauche (_, (_, _, ori)) = ori := (!ori + 1) mod 4

let droite (_, (_, _, ori)) = ori := (!ori + 3) mod 4
```

Question P7 :

```
let avancer (grille, (l, c, ori)) =
  let _ = match !ori with
  | 0 -> incr c; if !c = Array.length grille.(0) || fst grille.(!l).(!c) = 0 then failwith "Game over !"
  | 1 -> decr l; if !l = -1 || fst grille.(!l).(!c) = 0 then failwith "Game over !"
  | 2 -> decr c; if !c = -1 || fst grille.(!l).(!c) = 0 then failwith "Game over !"
  | 3 -> incr l; if !l = Array.length grille || fst grille.(!l).(!c) = 0 then failwith "Game over !"
  | _ -> failwith "Orientation impossible"
  in if snd grille.(!l).(!c) then grille.(!l).(!c) <- (fst grille.(!l).(!c), false);
  if fini grille then failwith "Victoire !"
```

Question P8 :

```
let tableaux_to_fonctions t0 t1 =
  let grille, (l, c, ori) = instance in
  let rec f0 () = Array.iter traitement t0
  and f1 () = Array.iter traitement t1
  and traitement (cmd, coul) =
    let couleur = fst grille.(!l).(!c) in if coul = 0 || coul = couleur
    then match cmd with
    | -1 -> avancer instance
    | -2 -> gauche instance
    | -3 -> droite instance
    | 0 -> f0 ()
    | 1 -> f1 ()
    | _ -> failwith "Cas impossible avec deux tableaux !"
  in f0, f1
```

Question P9 :

La liste est forcément `[(0, 0)]` au début.

Question P10 :

```
let copie_instance (grille, (l, c, ori)) = (Array.map Array.copy) grille, (ref !l, ref !c, ref !ori);;
```

Question P11 :

```
let verifie instance soumission =
  let inst = copie_instance instance in
  let rec mouline l = match l with
  | [] -> false
  | (i, j)::q when j = Array.length soumission.(i) -> mouline q
  | (i, j)::q ->
    begin
      let grille, (l, c, _) = inst in
      let couleur = fst grille.(!l).(!c) in
      let (cmd, coul) = soumission.(i).(j) in
      if coul <> 0 && coul <> couleur then mouline ((i, j+1)::q)
      else match cmd with
      | -1 -> avancer inst; mouline ((i, j+1)::q)
      | -2 -> gauche inst; mouline ((i, j+1)::q)
      | -3 -> droite inst; mouline ((i, j+1)::q)
      | ind -> mouline ((ind, 0)::(i, j+1)::q)
    end
  in try mouline [(0, 0)] with Failure e -> e = "Victoire !"
```

Question P12 :

F1 : ↶ F2 ↶ ↑ F1
F2 : ↑ F2 ↶ ↶ ↑

D'après le site, il existe une solution avec neuf instructions.

Question P13 : En attente de quelqu'un qui me proposera une solution, flemme de réfléchir à nouveau...