```
type marque = { c : bool; p : int*int}
type grille = marque list
(** Ex1 *)
(* Ex1.1 *)
let g1 = [{c = true; p = (0,0)};
          {c = true; p = (0,1)};
          {c = true; p = (0,2)};
          {c = false; p = (1,1)};
          {c = false; p = (2,0)};
          {c = false; p = (2,2)}]

(* Ex 1.2 *)
let dans_le_bornes size =
    List.for_all
      (fun m ->
        let row = fst m.p in
        let col = snd m.p in
        row >= 0 && row < size && col >= 0 && col < size)
    g1

let dans_le_bornes_2 size =
    List.for_all
      (fun m ->
        let row,col = m.p in
        row >= 0 && row < size && col >= 0 && col < size)
    g1

let dans_le_bornes_3 size =
    List.for_all
      (fun { p; _ } ->
        let row,col = p in
        row >= 0 && row < size && col >= 0 && col < size)
    g1

let dans_le_bornes_4 size =
    List.for_all
      (fun { p = (row,col); _ } ->
      row >= 0 && row < size && col >= 0 && col < size)
    g1

let dans_le_bornes_bis l =
  List.for_all
    (fun { p = (x, y) } -> 0 <= x && x <= 2  &&  0 <= y && y <= 2) l

(* Ex 1.3 *)
let existe_symbole g i j =
    List.exists
      (fun m ->
        let row = fst m.p in
        let col = snd m.p in
        i = row && j = col)
    g

let existe_symbole_2 g i j =
    List.exists
      (fun m ->
        let row,col = m.p in
        i = row && j=col)
    g

let existe_symbole_3  g i j =
    List.exists
      (fun { p; _ } ->
        let row,col = p in
        i = row && j = col)
    g
```

```
let existe_symbole_4 g i j =
    List.exists
      (fun { p = (row,col); _ } -> i = row && j = col)
    g

(* Ex 1.4 *)
let rec sans_doublons l =
    match l with
     | [] -> true
     | x::s ->
        let row = fst x.p in
        let col = snd x.p in
        not (existe_symbole l row col) && sans_doublons l

let rec sans_doublons grid =
    match grid with
      | [] -> true
      | { p = (row,col); _ } :: rest ->
        not (existe_symbole rest row col) && sans_doublons rest

(* Ex 1.5 *)
let compter grid =
    List.fold_left
      (fun (x,o) mark ->
        if mark.c then (x+1,o) else (x,o+1))
        (0,0) grid

let compter_bis l =
  List.fold_left (fun (x, o) { c = b } ->
    if b then (x + 1, o) else (x, o + 1)) (0, 0) l

(* Ex 1.6 *)
let bonne_grille grid =
    dans_le_bornes 3 && sans_doublons grid &&
    let (x,o) = compter grid in
    x-o = 1 || x-o = 0

let bonne_grille_bis l =
  let (cr, ce) = compter l in
   sans_doublons l && dans_le_bornes_bis l  &&  (cr = ce || cr = ce + 1)

exception Invalid_grid
(* Ex 1.7 *)
let bonne_grille_exn grid =
    if not(bonne_grille grid) then raise Invalid_grid
(* Ex 1.8 *)
let gagne l =
  let m = existe_symbole l in
  let horiz i = m i 0 && m i 1 && m i 2 in
  let vert j = m 0 j && m 1 j && m 2 j in
  let diag1 = m 0 0 && m 1 1 && m 2 2 in
  let diag2 = m 0 2 && m 1 1 && m 2 0 in
  horiz 0 || horiz 1 ||  horiz 2 ||
  vert 0 ||  vert 1 ||  vert 2 || diag1 || diag2
(* Ex 1.8 alter *)
let list_init f n =
    let rec aux acc n =
      if n < 0 then acc
      else aux (f n :: acc)(n-1)
    in
aux [] (n-1)
;;
(* 所有可能赢的情况 *)
let all_winning_grid n =
```

```
    list_init (fun i -> i,i) n
    :: list_init (fun i -> i,n-i-1) n
    :: (list_init (fun i -> list_init (fun j -> i,j) n) n
     @  list_init (fun i -> list_init (fun j -> j,i) n) n)
;;
let winning_grid grid =
    List.exists
      (fun list ->
        List.for_all
          (fun (row,col) -> existe_symbole grid row col)
          list)
    (all_winning_grid 3)
;;
(* Ex 1.9 *)
let extraire grid s = List.filter (fun { c; _ } -> c = s) grid;;
let extraire_bis l s = List.filter (fun m -> m.c = s) l

(* Ex 1.10 *)
let qui_gagne grid =
    try
      bonne_grille_exn grid;
      if winning_grid (extraire grid true) then print_endline "X win"
      else if winning_grid (extraire grid false) then print_endline "O win"
      else print_endline "Nul..."
    with
      Invalid_grid -> print_endline "Invalid_grid"
let qui_gagne_bis l =
  let msg =
    try
      bonne_grille_exn l;
      let croix = extraire l true in
      let ronds = extraire l false in
      if gagne croix then "les croix gagnent"
      else if gagne ronds then  "les ronds gagnent"
      else "partie nulle"
    with | Invalid_grid -> "Grille invalide"
  in
  Printf.printf "%s\n" msg

(** N Reines *)
let list_of_int n =
  let rec loi acc n =
  if n <= 0 then acc
  else loi (n::acc) (n-1)
  in
  loi [] n

let succl = List.map succ
let predl = List.map pred

let diff a b =
  List.fold_left (fun acc x -> if List.mem x b then acc else x::acc) [] a

let remove a x =
  List.fold_left (fun acc y -> if y = x then acc else y :: acc) [] a

let rec queens a b c =
  if a=[] then 1
  else
    let e = diff (diff a b) c in
    List.fold_left
      (fun acc d ->
         acc + queens (remove a d) (succl (d::b)) (predl (d::c))) 0 e

let queens n = queens (list_of_int n) [] []
```