Corrigé feuille d'exercices n°14 de l'option d'informatique.

```
let n = 10;;           (*pour un damier 10x10 *)

(* fonction de lecture du fichier contenant le labyrinthe *)
(* lit_labyrinthe : string -> char vect vect = <fun> *)
let lit_labyrinthe nom_fichier =
   let lab = make_matrix (n+2) (n+2) '1'
         and fichier = open_in nom_fichier and ligne = ref("") in
               for i=1 to n do
                     ligne := input_line fichier;
                     for j=1 to n do lab.(i).(j) <- !ligne.[j-1] done;
               done;
   lab;;

(* fonction d'affichage d'un labyrinthe *)
(* imprime : char vect vect -> unit = <fun> *)
let imprime lab =
      for i=1 to n do
            for j=1 to n do print_char(lab.(i).(j)) done;
            print_newline();
     done;;

(* fonction pour modifier un élément d'un labyrinthe *)
(* modifie : 'a vect vect -> int -> int -> 'a -> unit = <fun> *)
let modifie lab i j c = lab.(i).(j) <- c;;

(* construction d'une pile à partir d'une liste référencée *)
let pile = ref [];;  (* pile : '_a list ref = ref [] *)

(* fonction pour ajouter un élément à une pile *)
(* empile : 'a list ref -> 'a -> unit = <fun> *)
let empile p x =
      p := x::(!p);;

(* fonction pour retirer un élément à une pile *)
(* depile : 'a list ref -> 'a = <fun> *)
let depile p =
      match (!p) with
     [] -> failwith "depile : Erreur la pile est vide"
   | x::xs -> p := xs; x;;
```

Fonction qui ajoute à la pile les cases accessibles à partir de la case (i, j) c'est-à-dire les case adjacentes libres dans les directions N, E,S, O.
(`directions : 'a -> char vect vect -> int -> int -> unit = <fun>`)

```
let directions p lab i j =
let test lab x y = (lab.(x).(y) = '0' || lab.(x).(y) == ' ') in
     if test lab (i-1) (j)  then empile p (i-1,j);
     if test lab (i+1) (j)  then empile p (i+1,j);
     if test lab (i) (j-1)  then empile p (i,j-1);
     if test lab (i) (j+1)  then empile p (i,j+1);;
```

Fonction itérative pour trouver un chemin dans un labyrinthe conduisant à la sortie.
(`sortie_trouvée : char vect vect -> int -> int -> bool = <fun>`)

```
    let sortie_trouvée lab i0 j0 =
48        let i = ref(i0) and j = ref(j0) in
                directions pile lab (i0) (j0);
50              while !pile <> [] && lab.(!i).(!j) <> ' ' do
                      lab.(!i).(!j) <- 'V';
52                    let (ii,jj) = depile pile in
                            i := ii; j := jj;
54                          directions pile lab (!i) (!j)
                        done;
56        lab.(!i).(!j) = ' ';;

58  (* essai : char vect vect -> int -> int -> unit = <fun> *)
    let essai lab i0 j0 =
60        if lab.(i0).(j0) = '1' or lab.(i0).(j0) = 'M'
            then print_string("Point de départ non permis")
62          else
               begin
64                 if sortie_trouvée lab i0 j0
                           then print_string("J'ai trouvé la sortie")
66                         else print_string("Je suis enfermé");
                   print_newline();
68                 imprime lab;
                end;;
70
    let lab = lit_labyrinthe "labyr.dta";;
72  imprime lab;;
    essai lab 9 8;;
```

```
J'ai trouvé la sortie
MXXMO MOMO
OMXMXXMMOM
OMXMXMMMMO
MMXXXXXMOM
OMMXXMXXXX
OMOMXMXMXM
MMOOMXXMXM
OMOOMMMXXX
OMMOMMXXXM
MOOMMXXMXM
- : unit = ()
```

Version récursive avec matérialisation du chemin conduisant vers la sortie.

```
74  let sortie_trouvée lab i0 j0 =
        let rec cherche i j =
76          match lab.(i).(j) with
              ' ' -> true
78          | 'O' -> lab.(i).(j) <- 'V';
                     if cherche (i+1) (j) then lab.(i).(j) <- '+'
80              else if cherche (i-1) (j) then lab.(i).(j) <- '+'
                else if cherche (i) (j+1) then lab.(i).(j) <- '+'
82              else if cherche (i) (j-1) then lab.(i).(j) <- '+';
                     if lab.(i).(j) = '+' then true else false
84          | _ -> false
         in
```

```
86   cherche i0 j0;;
```

Dans ce cas, on peut voir le chemin vers la sortie ainsi que toutes les cases qui ont été explorées :

```
#J'ai trouvé la sortie
MVVM+ MOMO
OMVM+OMMOM
OMVM+MMMMO
MMVV+++MVM
OMMVVM+++V
OMOMVMVM+M
MMOOMVVM+M
OMOOMMM++O
OMMOMMO+VM
MOOMMOOMVM
- : unit = ()
```

`dessine_chemin` : fonction pour dessiner le chemin vers la sortie si celui-ci existe.
`( dessine_chemin : char vect vect -> (int * int) list -> unit = <fun> )`

```
    let rec dessine_chemin lab = function
88    [] -> ()
    | (i,j)::xs -> lab.(i).(j) <- '+'; dessine_chemin lab xs;;
```

Ici,on empile les différents chemins issus de la position de départ menant aux cases accessibles depuis la case (i,j).
(`Directions : (int * int) list list ref -> char vect vect -> int -> int -> (int * int) list -> unit = <fun>`)

```
90   let Directions p lab i j l =
     let test lab x y = (lab.(x).(y) = 'O' || lab.(x).(y) == ' ') in
92        if test lab (i-1) (j)  then empile p ((i-1,j)::l);
          if test lab (i+1) (j)  then empile p ((i+1,j)::l);
94        if test lab (i) (j-1)  then empile p ((i,j-1)::l);
          if test lab (i) (j+1)  then empile p ((i,j+1)::l);;
```

Version itérative avec tracé d'un chemin vers la sortie.
(`sortie_trouvée : char vect vect -> int -> int -> bool = <fun>`)

```
96   let sortie_trouvée lab i0 j0 =
          let i = ref(i0) and j = ref(j0) and l = ref [] and pile = ref [] in
98              Directions pile lab (i0) (j0) [i0,j0];
                while !pile <> [] && lab.(!i).(!j) <> ' ' do
100                    lab.(!i).(!j) <- 'X';
                       l := depile pile;
102                          match !l with
                             | (ii,jj)::xs -> i := ii; j := jj;
104                                              Directions pile lab (ii) (jj) (!l)
                             | _ -> failwith "Erreur : un chemin est vide"
106                   done;
          if lab.(!i).(!j) = ' '
108               then begin dessine_chemin lab !l; true end
                  else false;;
```