```
type 'a btree = Nil | Node of 'a * 'a btree * 'a btree ;;

  (* exercice 1 *)

let rec size = function
  | Nil              -> 0
  | Node (_, fg, fd) -> 1 + size fg + size fd ;;


let rec leaf = function
  | Nil                -> 0
  | Node (_, Nil, Nil) -> 1
  | Node (_, fg, fd)   -> leaf fg + leaf fd ;;


let rec member x = function
  | Nil              -> false
  | Node (y, fg, fd) -> x = y || member x fg || member x fd ;;


let rec height = function
  | Nil              -> -1
  | Node (_, fg, fd) -> 1 + max (height fg) (height fd) ;;


  (* exercice 2 *)

let rec tag_prefix = function
  | Nil              -> []
  | Node (x, fg, fd) -> x::(tag_prefix fg) @ (tag_prefix fd) ;;


let rec tag_infix = function
  | Nil              -> []
  | Node (x, fg, fd) -> (tag_infix fg) @ (x::(tag_infix fd)) ;;


let rec tag_suffix = function
  | Nil              -> []
  | Node (x, fg, fd) -> (tag_suffix fg) @ (tag_suffix fd) @ [x] ;;


  (* exercice 3 *)

let rec map_tree f = function
  | Nil              -> Nil
  | Node (x, fg, fd) -> Node (f x, map_tree f fg, map_tree f fd) ;;


let rec fold_tree f t b = match t with
  | Nil              -> b
  | Node (a, fg, fd) -> f a (fold_tree f fg b) (fold_tree f fd b) ;;


let size2 t = fold_tree (fun x y z -> 1 + y + z) t 0 ;;
let height2 t = fold_tree (fun x y z -> 1 + max y z) t (-1) ;;
let tag_prefix2 t = fold_tree (fun x y z -> x::y @ z) t [] ;;
let tag_infix2 t = fold_tree (fun x y z -> y @ x::z) t [] ;;
let tag_suffix2 t = fold_tree (fun x y z -> y @ z @ [x]) t [] ;;
```

```
  (* exercice 4 *)

let rec miroir t1 t2 = match (t1, t2) with
  | Nil, Nil                                -> true
  | Nil, _                                  -> false
  | _, Nil                                  -> false
  | Node (x, fg1, fd1), Node (y, fg2, fd2) -> x = y && miroir fg1 fd2 &&
miroir fd1 fg2 ;;

let symmetric = function
      | Nil               -> true
      | Node (_, fg, fd) -> miroir fg fd ;;


type 'a ntree = Nil | Node of 'a * ('a ntree list) ;;

  (* exercice 5 *)

let rec size = function
  | Nil             -> 0
  | Node (_, fils) -> it_list (fun a b -> a + (size b)) 1 fils ;;


let rec member x = function
  | Nil -> false
  | Node (y, fils) -> x = y || exists (member x) fils ;;


let rec height = function
  | Nil -> -1
  | Node (_, fils) -> 1 + it_list (fun a b -> max a (height b)) (-1) fils
;;


let rec sum = function
  | Nil -> 0
  | Node (x, fils) -> it_list (fun a b -> a + (sum b)) x fils ;;
```