```
(************************************************************************)
(*                    Arbres binaires de recherche                      *)
(************************************************************************)


type 'a arbre = Feuille
              | Noeud   of 'a arbre * 'a * 'a arbre
;;


(* recherche : 'a arbre -> 'a -> bool *)

let recherche a e =
  let rec cherche = function
     Noeud (g, e', d) -> if e=e' then
                           true
                         else
                           if e<e' then cherche g else cherche d
   | Feuille          -> false

  in cherche a
;;


(* insertion : 'a arbre -> 'a -> 'a arbre *)

let insertion a e =
  let rec insere = function
     Noeud (g, e', d) -> if e<e' then Noeud(insere g, e', d)
                                 else Noeud(g, e', insere d)
   | Feuille          -> Noeud(Feuille, e, Feuille)

  in insere a
;;


(* construit : 'a list -> 'a arbre *)

let rec construit = function
    []   -> Feuille
  | e::l -> insertion (construit  l) e
;;


(* parcours : 'a arbre -> 'a list *)

let rec parcours = function
    Feuille        -> []
  | Noeud(g, e, d) -> (parcours g)@(e::(parcours d))
;;


(* suppression : 'a arbre -> 'a -> 'a arbre *)

let suppression a e =
  let rec plus_petit = function
     | Noeud(Feuille, e', d) -> e',d
     | Noeud(g, e', d)       -> let e'',g' = plus_petit g in
                                e'', Noeud(g', e', d)
  in
  let sup_racine = function
     | Noeud(Feuille ,_, d) -> d
     | Noeud(g, _, Feuille) -> g
```



```
     | Noeud(g, e, d)       ->
            let p,d' = plus_petit d in
            Noeud(g, p, d')
  in
  let rec descente = function
      Feuille                 -> Feuille
    | (Noeud(g, e', d)) as a -> if e=e' then
                                  sup_racine a
                                else
                                  if e<e' then Noeud(descente g, e', d)
                                          else Noeud(g, e', descente d)
  in descente a
;;
```