```
let rec list_map f = function
  | [] -> []
  | h::t -> (f h)::(list_map f t);;

let rec list_select p = function
  | [] -> []
  | h::t -> if p h then h::(list_select p t) else list_select p t;;

let rec insere e = function
  | [] -> [e]
  | h::t as l -> if e < h then e::l else h::(insere e t);;

let rec tri_insertion = function
  | [] -> []
  | h::t -> let l = tri_insertion t in insere h l;;

let rec tri_rapide list =
  let rec split pivot = function
    | [] -> [],[]
    | h::t -> let l1,l2 = split pivot t in
        if h < pivot then (h::l1,l2) else (l1,h::l2)
  in match list with
    | [] -> []
    | pivot::suite -> let l1,l2 = split pivot suite in
        (tri_rapide l1)@(pivot::(tri_rapide l2)) ;;

let list_rev list =
  let rec boucle accu = function
    | [] -> accu
    | h::t -> boucle (h::accu) t
    in boucle [] list ;;

let rec puissance x n =
  if n = 0 then 1
  else let y = puissance x (n/2) in
       if n mod 2 = 0 then y * y else y * y * x;;

let rec pgcd x y = if y = 0 then x else pgcd y (x mod y) ;;

let rec base x b =
  if x = 0 then []
  else (x mod b)::(base (x/b) b) ;;

let rec eval_base b = function
  | [] -> 0
  | h::t -> h + b*(eval_base b t) ;;

let addition b x y =
  let rec boucle x y r = match (x,y) with
    | [],[] -> if r = 0 then [] else [1]
    | hx::tx,[] -> let vl = hx + r in (vl mod b)::(if vl < b then tx else boucle tx [] 1)
    | [],hy::ty -> let vl = hy + r in (vl mod b)::(if vl < b then ty else boucle [] ty 1)
    | hx::tx,hy::ty -> let vl = hx + hy + r in
      (vl mod b)::(boucle tx ty (if vl < b then 0 else 1))
  in boucle x y 0;;

let fibo n =
  let rec boucle u v n =
    if n = 0 then u else boucle v (u+v) (n-1)
  in boucle 0 1 n ;;

let derive poly =
  let rec boucle k = function
  | [] -> []
  | h::t -> (k*h)::(boucle (k+1) t)
  in match poly with
    | ([] | [_]) -> []
    | _::t -> boucle 1 t ;;
```

```
let recherche e tab =
  let n = vect_length tab in
  let rec boucle k =
      (k < n) && (tab.(k) = e || boucle (k+1))
  in boucle 0 ;;

let recherche_dicho e tab =
  let n = vect_length tab in
  let rec boucle i j =
      if i >= j-1 then (tab.(i) = e || tab.(j) = e)
      else let k = (i + j) / 2 in
        if tab.(k) > e then boucle i k
        else boucle k j
  in boucle 0 (n-1) ;;

type arbre = Feuille of int | Noeud of arbre * arbre * int ;;

let rec max_min = function
  | Feuille vl -> vl,vl
  | Noeud(ag,ad,vl) -> let max_g,min_g = max_min ag and max_d,min_d = max_min ad in
                       (max max_g (max max_d vl), min min_g (min min_d vl)) ;;

let abr arbre =
  let rec boucle mini maxi = function
  | Feuille vl -> (mini < vl) && (vl < maxi)
  | Noeud(ag,ad,vl) -> (mini < vl) && (vl < maxi)
      && (boucle mini vl ag) && (boucle mini maxi ad)
  in let maxi,mini = max_min arbre in boucle (mini-1) (maxi+1) arbre ;;

let rec cherche_abr e = function
  | Feuille vl -> vl = e
  | Noeud(ag,ad,vl) -> (vl = e) || (vl > e && cherche_abr e ag) || (vl < e && cherche_abr e ad) ;;

let rec max_sum = function
  | Feuille vl -> vl
  | Noeud(ag,ad,vl) -> vl + max (max_sum ag) (max_sum ad) ;;

let hierarchie arbre =
  let rec boucle accu = match accu with
    | [] -> []
    | (Feuille vl)::t -> vl::(boucle t)
    | (Noeud(ag,ad,vl))::t -> vl::(boucle (t @ [ag;ad]))
  in boucle [arbre] ;;

type arbre_expr = Value of int | Op_bin of arbre_expr * arbre_expr | Op_un of arbre_expr ;;

let rec prefixe = function
  | Value vl -> string_of_int vl
  | Op_bin(eg,ed) -> "op_bin ("^(prefixe eg)^","^(prefixe ed)^")"
  | Op_un(e) -> "op_un ("^(prefixe e)^")" ;;

let rec infixe = function
  | Value vl -> string_of_int vl
  | Op_bin(eg,ed) -> "("^(infixe eg)^") op_bin ("^(infixe ed)^")"
  | Op_un(e) -> "op_un ("^(infixe e)^")" ;;

let rec suffixe = function
  | Value vl -> string_of_int vl
  | Op_bin(eg,ed) -> "("^(suffixe eg)^","^(suffixe ed)^") op_bin"
  | Op_un(e) -> "("^(suffixe e)^" op_un)" ;;

let accessible tab_succ s =
  let n = vect_length tab_succ in
  let tab_acc = make_vect n false in
  let rec boucle = function
    | [] -> ()
    | h::t -> if not tab_acc.(h) then (tab_acc.(h) <- true ; boucle tab_succ.(h)) ; boucle t
  in boucle [s] ; tab_acc ;;
```