Corrigé Colle d'info n°3

1 - Représentation par listes

```
type 'a dico == 'a list ref;;

let dic = ref [];;
```

Exercice n°1 : La fonction «`vérif`» retourne `true` si le dictionnaire ne contient pas de doublons et `false` sinon.

```
let vérif dict =
   let rec est = fun
      x [] -> false
     | x (y::ys) when x=y -> true
     | x (y::ys) -> est x ys in
   let rec Vérif = function
       [] -> true
     | x::xs when est x xs -> false
     | x::xs -> Vérif xs  in
Vérif !dict;;
```

Exercice n°2 : La fonction «`nettoie`» élimine les doublons dans le dictionnaire `dict`.

```
let nettoie dict =
  let rec élimine = fun
       x [] -> []
    | x (y::ys) when x=y -> élimine x ys
    | x (y::ys) -> y::(élimine x ys) in
  let rec Nettoie = function
      [] -> []
    | x::xs  -> x::(Nettoie (élimine x xs)) in
dict := Nettoie !dict;;
```

Exercice n°3 : la fonction «`contient`» regarde si une clé **c** est dans un dictionnaire.

```
let rec contient c = function
  [] -> false
| x::xs when c=x -> true
| x::xs -> contient c xs;;
```

Exercice n°4 : La fonction «`insère`» insère une clé dans un dictionnaire dans elle n'y est pas déjà présente.

```
let insére clé dic =
let rec Insère = fun
 x [] -> [x]
| x (y::ys) when x=y -> y::ys
| x (y::ys) -> y::(Insère x ys) in
dic := Insère clé !dic;;
```

2 - Représentation triée

```
let tmax = 100;;

let dico = make_vect tmax "";;

let longueur dic =
let i = ref(0) in
while dic.(!i) <> "" do incr i done;
!i;;
```

Exercice 5 :

```
exception désordre;;

let vérif2 dic =
try
let l = longueur dico in
for i = 0 to l-2 do
    if dic.(i) >= dic.(i+1) then raise désordre
done;
true
with désordre -> false;;
```

Exercice n°6 :

```
let recherche c dico =
  let l = longueur dico in
    let rec trouve c q r =
       if (r-q < 0)
           then (false,min r (l-1))
           else
              let s = (q+r)/2 in
              let d = dico.(s) in
                  match (c,d) with
                    (c,d) when c = d -> (true,s)
                  | (c,d) when c < d -> trouve c q (s-1)
                  | (c,d) -> trouve c (s+1) r
       in
  if l=0 then (false,-1)
         else if c < dico.(0) then (false,-1)
                              else trouve c 0 l;;
```

Exercice n° 7 :

```
let insère2 c dico =
let (bool,pos) = recherche c dico in
   if not(bool) then
      begin
      for i = longueur dico downto max 1 (pos+1) do
            dico.(i) <- dico.(i-1);
            done;
      dico.(pos+1) <- c
      end;;
```

3 - Représentation arborescente

Exercice n°8 :

```
type 'a dictionnaire = vide
 | noeud of 'a dictionnaire * 'a * 'a dictionnaire;;

let rec contient3 c = function
   vide -> false
| noeud(fg,d,fd) when c=d -> true
| noeud(fg,d,fd) when c<d -> contient3 c fg
| noeud(fg,d,fd) -> contient3 c fd;;
```

Exercice n°9 :

```
let rec insère3 c dico =
     let rec insère c = function
         vide -> noeud(vide,c,vide)
    | (noeud(fg,d,fd) as arbre) when c=d -> arbre
    | noeud(fg,d,fd) when c<d -> noeud(insère c fg,d,fd)
    | noeud(fg,d,fd) -> noeud(fg,d,insère c fd)
in dico := insère c !dico;;
```

Exercice n°10 :

```
let rec ajoute3 liste dico =
match liste with
    [] -> ()
| x::xs -> insère3 x dico; ajoute3 xs dico;;

ajoute3 [18;42;3;14;29;61;15;45;60;9] dico;;
```

Exercice n°11 :

```
let rec parcours = function
  vide -> []
| noeud(fg,d,fd) -> (parcours fg)@[d]@(parcours fd);;
```

Exercice n°12 :

```
let rec max_dico = function
 vide -> failwith "dictionnaire vide"
 | noeud (_,x,vide) -> x
 | noeud(_,x,fd) -> max_dico fd;;

 let rec min_dico = function
  vide -> failwith "dictionnaire vide"
  | noeud (vide,x,_) -> x
  | noeud(fg,x,_) -> min_dico fg;;

let alea_bool () = random__int 2 = 0;;

(* pour vérifier que la probabilté est voisine de 1/2 *)
let x = ref 0. and n = 1000 in
for i = 0 to n do
  if alea_bool () then x := !x +. 1.;
done;
print_float (!x /. (float_of_int n));;

let rec supprime x = function
  vide -> vide
| noeud(vide,y,fd) when y=x -> fd
| noeud(fg,y,vide) when y=x -> fg
| noeud(fg,y,fd) when x=y ->
  if alea_bool() then (let z = max_dico fg in noeud(supprime z fg,z,fd))
           else (let z = min_dico fd in noeud(fg,z,supprime z fd))
| noeud(fg,y,fd) (* x<>y *) ->
  if x< y then noeud(supprime x fg,y,fd)
      else noeud(fg,y,supprime x fd);;
```

Exercice n°13 :

```
let rec profondeur x = function
   vide -> -1
 | noeud(fg,y,fd) when x=y -> 0
 | noeud(fg,y,fd) when x < y ->
    let p = profondeur x fg in
    if  p = -1 then -1 else p+1;
 | noeud(fg,y,fd) (* when x > y *) ->
   let p = profondeur x fd in
     if  p = -1 then -1 else p+1;;
```

Exercice n°14 :

```
let rec contient3 x = function
  vide -> false
| noeud(_,y,_) when x=y -> true
| noeud(fg,y,fd) (* when x<>y *) ->
   if x < y then contient3 x fg
         else contient3 x fd;;

let rec est_descendant_de m1 m2 = function
   vide -> false
| noeud(fg,y,fd) when y=m1 ->
  if m2 <m1 then contient3 m2 fg
        else contient3 m2 fd
| noeud(fg,y,fd) (* when y<>m1 *)
  if m1 < y then est_descendant_de m1 m2 fg
        else est_descendant_de m1 m2 fd;;
```

Exercice n°15 :

```
let ancetre x y tree =
let rec aux = function
  vide -> failwith "erreur";
| noeud(fg,z,fd)  when (x=z || z=y)   -> z
| noeud(fg,z,fd)  when (x<z && z<y) || (y<z && z<y) ->z
| noeud(fg,z,_)   when (x<z && y<z)   -> aux fg
| noeud(_,z,fd) (*when (x>z && y>z)*) -> aux fd
in
  if (contient3 x tree) && (contient3 y tree)
     then aux tree
   else failwith "l'un des mots n'est pas dans le dico";;
```

Exercice n°16 : La sous-fonction `aux` s'applique à des arbres et retourne un triplet constitué d'un booléen égal à `true` si l'arbre est un arbre binaire de recherche (et `false` sinon) et de deux entiers représentant le plus petit et le plus grand élément de l'arbre.

```
let verif_dico tree =
let rec aux = function
  vide -> failwith "erreur"
| noeud(vide,x,vide) -> (true,x,x)
| noeud(vide,x,fd) -> let (b,m,M) = aux fd in
                (b && m>x,x,M)
| noeud(fg,x,vide) -> let (b,m,M) = aux fg in
                (b && M<x,m,x)
| noeud(fg,x,fd) -> let (b,m,M) = aux fg and (b',m',M') = aux fd in
                (b && M<x && m'>x,m,M')
in
 let (b,m,M) = aux tree in
    b;;
```