Corrigé feuille d'exercices d'informatique n°8

1 arbres

1. (* Déclaration du type arbre ternaire *)

```
type 'a arbre_ternaire =
    Vide
| feuille of 'a
| noeud of 'a * 'a arbre_ternaire * 'a arbre_ternaire * 'a arbre_ternaire;;
```

(* construction de l'arbre donné en exemple dans l'énoncé *)

```
let a1 = noeud(9,feuille(14),feuille(5),feuille(10));;
let a2 = noeud(10,feuille(25),a1,feuille(2));;
let a3 = noeud(15,feuille(11),feuille(8),feuille(9));;
let a4 = noeud(5,feuille(7),feuille(10),a3);;
let arbre = noeud(15,a2,feuille(12),a4);;
```

(* Déclaration du type file d'attente *)

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

2. (* fonction pour ajouter un élément à un liste d'attente *)
(* ajoute : 'a -> 'a file -> unit = <fun> *)

```
let ajoute x (f:'a file) =
    f := x::(!f);;
```

(* fonction pour retirer le premier élément à une liste d'attente *)
(* sortie : 'a arbre_ternaire file -> 'a arbre_ternaire = <fun> *)

```
let sortie (f:'a file) =
     match (rev !f) with
        [] -> Vide
    |  x::xs -> f := rev(xs); x;;
```

(* fonction de création de liste d'attente *)
(* crée_file : 'a -> 'b file = <fun> *)

```
let crée_file a =
   (ref( []):'a file);;
```

Réécriture des fonctions précédentes en utilisant un tableau et deux "pointeurs" :

```
let taille_max = 100;;

type 'a file = {contenu : 'a vect; taille : int; mutable début : int; mutable fin : int};;

let crée_file a = {contenu = make_vect taille_max a; taille = taille_max; début = 0; fin = 0};;

let ajoute x f =
      let nouvelle_fin = (f.fin + 1) mod f.taille in
            if nouvelle_fin = f.début then failwith "ajoute : débordement de la file";
            f.contenu.(f.fin) <- x;
            f.fin <- nouvelle_fin;;
```

```
let sortie f =
      let nouveau_début = (f.début + 1) mod f.taille
           and x = f.contenu.(f.début) in
                try
             if f.fin = f.début then failwith "sortie : file vide";
                 f.début <- nouveau_début;
                  x
                  with Failure texte -> Vide;;
```

3. (* fonction qui imprime les informations d'un arbre ternaire dans l'ordre hiérarchique *)
(* `hiérarchique : int arbre_ternaire -> unit = <fun>` *)

```
let hiérarchique a =
   let file = crée_file Vide in
       ajoute a file;
       let rec parcours f =
          match sortie(f) with
            Vide -> ()
          | feuille(x) -> print_int x; print_string " "; parcours f
          | noeud(x,a1,a2,a3) -> print_int x;print_string " ";
                                 ajoute a1 f; ajoute a2 f; ajoute a3 f;
                                 parcours f
       in
   parcours file;;

hiérarchique arbre;;
```

2 automates

(* Déclarations des types *)

```
type état == int;;
type ÉtatsListe == état list;;

let rebut = (-1 : état);;
let début = (0 : état);;

type caractère == char;;
type transition == (état*caractère)*état;;
type TransitionListe == transition list;;
```

4. (* fonction pour convertir un mot en la liste de ses caractères *)
(* `liste_de_mot : string -> char list = <fun>` *)

```
let liste_de_mot mot =
   let l = ref([]) in
       for i = string_length mot downto 1 do
           l := mot.[i-1]::(!l);
           done;
       !l;;
```

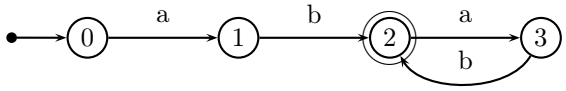
5. (* fonction de transition *)
(* f_transition : état * caractère -> automate -> état = <fun> *)

```
let f_transition (e:(état*caractère)) (a:TransitionListe) =
    try   assoc e a
  with Not_found -> rebut;;
```

6. (* fonction qui vérifie si un mot appartient à un langage reconnu par un automate *)
(* vérif : état -> automate -> string -> bool = <fun> *)

```
let vérif (final : ÉtatsListe) (a:TransitionListe) mot =
    let Mot = liste_de_mot mot in
let rec analyse = function
    (e,[]) when mem e final -> true
  | (_,[]) -> false
  | (e,_) when e = rebut -> false
  | (e,x::xs) -> analyse(f_transition (e,x) a,xs)
in
   analyse (début,Mot);;
```

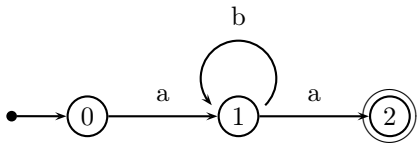
7. Définition d'un automate reconnaissant les mots de la forme "abab...ab" que l'on peut shématiquement représenter par :




```
let final1 = (2 : état);;
let automate1 = [((0,‘a‘),1); ((1,‘b‘),2);((2,‘a‘),3);((3,‘b‘),2)];;

vérif final1 automate1 "abababab";;
vérif final1 automate1 "ababababb";;
vérif final1 automate1 "abababa";;
vérif final1 automate1 "abc";;
```

8. Définition d'un automate reconnaissant les mots de la forme "ab...ba" que l'on peut shématiquement représenter par :




```
let final2 = (2 : état);;
let automate2 = [((0,‘a‘),1); ((1,‘b‘),1);((1,‘a‘),2)];;

vérif final2 automate2 "abbbbbbb";;
vérif final2 automate2 "aa";;
vérif final2 automate2 "abbbaba";;
vérif final2 automate2 "aca";;
vérif final2 automate2 "abbbbbbba";;
vérif final2 automate2 "aba";;
```