Corrigé du contrôle : scrutin de liste

Partie I. Listes paritaires

Question 1.

```
let rec composition = function
  | [] -> (0, 0)
  | t::q when t.Sexe = M -> let (a, b) = composition q in (a + 1, b)
  | _::q -> let (a, b) = composition q in (a, b + 1) ;;
```

Notons que la version suivante est aussi possible ; elle utilise une fonction auxiliaire dotée d'un accumulateur qui contient le nombre d'hommes et de femmes déjà rencontrés lors du parcours de la liste.

```
let composition =
  let rec aux (a, b) = function
    | [] -> (a, b)
    | t::q when t.Sexe = M -> aux (a + 1, b) q
    | _::q -> aux (a, b + 1) q
  in aux (0, 0) ;;
```

On peut aussi utiliser la fonctionnelle `it_list` :

```
let composition =
  let f (a, b) = function
    | c when c.Sexe = M -> (a + 1, b)
    | _ -> (a, b + 1) in
  it_list f (0, 0) ;;
```

Question 2.

```
let est_paritaire lst =
  let (a, b) = composition lst in a = b ;;
```

Question 3.

```
let rec enleveH = fun
  | _ [] -> []
  | 0 (t::q) when t.Sexe = M -> enleveH 0 q
  | n (t::q) when t.Sexe = M -> t::(enleveH (n - 1) q)
  | n (t::q) -> t::(enleveH n q) ;;
```

Question 4.

```
let rend_paritaire lst =
  let (_, b) = composition lst in enleveH b lst ;;
```

Partie II. Répartition des candidats

Question 5.

```
let positionH lst =
  let (_, n) = composition lst in
  let h = make_vect n 0 in
  let rec aux acc = function
    | [] -> h
    | t::q when t.Sexe = M -> aux (acc + 1) q
    | t::q -> h.(t.Id) <- acc ; aux acc q
  in aux 0 lst ;;
```

La fonction auxiliaire **aux** utilise un accumulateur qui dénombre les hommes rencontrés lors du parcours de la liste. À chaque fois que l'on rencontre une femme d'identité i, le tableau h est rempli avec la valeur de cet accumulateur.

Question 6.

```
let rec ajouterF a = fun
  | 0 lst -> {Sexe = F; Id = a}::lst
  | i (t::q) when t.Sexe = M -> t::(ajouterF a (i - 1) q)
  | i (t::q) -> t::(ajouterF a i q)
  | _ _ -> failwith "ajouterF" ;;
```

Compte tenu de l'hypothèse faite (la liste contient au moins i hommes), le dernier motif du filtrage est superflu ; il n'est présent que pour assurer l'exhaustivité du filtrage.

Question 7. On commence par définir une fonction construisant une liste constituée de n hommes :

```
let rec listeH = function
  | 0 -> []
  | n -> {Sexe = M; Id = n - 1}::(listeH (n-1)) ;;
```

ou si on préfère avec un accumulateur :

```
let listeH =
  let rec aux acc = function
    | 0 -> acc
    | n -> aux ({Sexe = M; Id = n - 1}::acc) (n - 1)
  in aux [] ;;
```

On définit ensuite la fonction principale :

```
let construire h =
  let n = vect_length h in
  let rec aux lst = function
    | k when k = n -> lst
    | k -> aux (ajouterF k h.(k) lst) (k + 1)
  in aux (listeH n) 0 ;;
```

♦ ♦
♦