Scrutin de liste

Durée : une heure

Un candidat est représenté par son sexe (masculin ou féminin) et son identité (on supposera pour simplifier qu'il s'agit d'un nombre entier). On définit donc le type `candidat` de la manière suivante :

```
type sexe = M | F ;;
type candidat = {Sexe : sexe ; Id : int} ;;
```

Pour définir un candidat, il faut lui attribuer un sexe et une identité, comme par exemple :

```
# let francois = {Sexe = M ; Id = 58} ;;
francois : candidat = {Sexe = M; Id = 58}
```

Rappelons que pour accéder aux caractéristiques d'un candidat ainsi défini, il faut faire suivre son nom d'un point (`.`) et de la caractéristique souhaitée :

```
# francois.Sexe ;;
- : sexe = M
# francois.Id ;;
- : int = 58
```

Une liste de candidats à une élection est représentée par une liste de type `candidat list`.

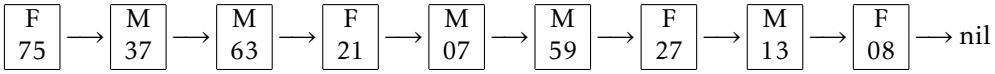


Figure 1 : un exemple de liste de candidats.

Partie I. Listes paritaires

Question 1. Rédiger en Caml une fonction `composition`, dont la signature est la suivante :

```
value composition : candidat list -> int * int
```

de sorte que `composition lst` renvoie le couple d'entiers (a, b) où a est le nombre d'hommes de la liste `lst` et b le nombre de femmes.

Question 2. En déduire une fonction `est_paritaire` déterminant si une liste contient autant d'hommes que de femmes.

```
value est_paritaire : candidat list -> bool
```

Question 3. On dispose d'une liste `lst` comportant un plus grand nombre d'hommes que de femmes, et on veut la rendre paritaire en supprimant certains hommes. Écrire en Caml une fonction `enleveH` de signature :

```
value enleveH : int -> candidat list -> candidat list
```

de sorte que `enleveH n lst` rend pour résultat une liste dans laquelle tous les hommes de la liste `lst` au-delà des n premiers ont étés supprimés, les autres candidats et candidates conservant les mêmes positions respectives.

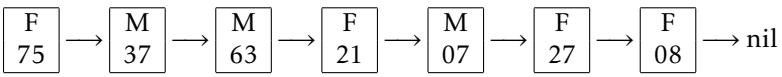


Figure 2 : la liste de la figure 1 après l'action de `enleveH 3 lst`.

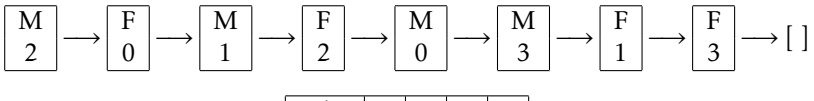
Question 4. Utiliser cette fonction dans une fonction :

```
value rend_paritaire : candidat list -> candidat list
```

qui renvoie une liste paritaire obtenue par suppression d'un certain nombre d'hommes situés à la fin de la liste initiale.

Partie II. Répartition des candidats

Dans toute la suite on note n le nombre de candidates qui figurent sur une liste ℓ, on suppose que deux candidats du même sexe ont des identités différentes et que les identités des candidates sont tous les nombres compris entre 0 et $n-1$. À chaque liste ℓ est associé un tableau de longueur n, que l'on appelle ici *tableau de positions* et que l'on note h_ℓ : pour tout $i \in [\![0, n-1]\!]$, $h_\ell(i)$ est le nombre d'hommes situés avant la candidate d'identité i dans la liste ℓ.



i	0	1	2	3
$h_\ell(i)$	1	4	2	4

Figure 3 : une liste paritaire, et le tableau de positions associé.

Rappelons qu'il y a deux façons de définir un tableau :
- en dressant la liste de ses éléments, séparés par un point-virgule et encadrés par les symboles `[|` et `|]` comme par exemple :

```
# let h = [|0; 4; 2; 4|] ;;
h : int vect = [|0; 4; 2; 4|]
```

- ou en créant un tableau à l'aide de la fonction `make_vect` en précisant sa taille et un élément initial puis en le remplissant ultérieurement.

```
value make_vect : int -> 'a -> 'a vect
```

Une fois créé, on peut consulter et modifier le contenu de ses cases : si `t` est un tableau et `k` un entier, `t.(k)` désigne le contenu de la case d'indice `k` (attention, dans un tableau de taille n, les indices s'échelonnent entre 0 et $n-1$), et on affecte la valeur `v` à la case d'indice `k` d'un tableau en suivant la syntaxe `t.(k) <- v`. Par exemple :

```
# let h = make_vect 4 1 ;;
h : int vect = [|1; 1; 1; 1|]
# h.(1) <- 4 ; h.(2) <- 2 ; h.(3) <- 4 ; h ;;
- : int vect = [|1; 4; 2; 4|]
```

Question 5. Écrire une fonction `positionH` de signature :

```
value positionH : candidat list -> int vect
```

qui calcule le tableau de positions h_ℓ d'une liste donnée ℓ.

Question 6. Rédiger une fonction de signature :

```
value ajouterF : int -> int -> candidat list ->  candidat list
```

de sorte que `ajouterF a i lst` renvoie une liste obtenue en ajoutant à la liste `lst` une candidate d'identité a dans une position telle qu'elle ait exactement i hommes placés avant elle. (On supposera donc que la liste `lst` contient au moins i hommes).

Question 7. En déduire une fonction de signature :

```
value construire : int vect -> candidat list
```

ayant pour objet, à partir d'un tableau h de longueur n, de construire une liste paritaire dont le tableau de positions est h. (On supposera donc que pour tout $i \in [\![0, n-1]\!]$, $h(i)$ est compris entre 0 et n.)

On rappelle que la fonction `vect_length` calcule la longueur d'un tableau.

♦ ♦
♦