

# Compte-rendu du projet Comment voter ?

Mathias Rivet

28 avril 2023

Les codes sources sont aussi disponible à l'adresse : <https://github.com/programindfr/INF201-Projet>

## Table des matières

<b>1</b>	<b>Scrutin uninominal</b>	<b>1</b>
1.1	Question 1	1
1.2	Question 2	1
1.3	Question 3	1
1.4	Question 4	2
1.5	Question 5	2
1.6	Question 6	2
1.7	Question 7	3
1.8	Question 8	3
1.9	Question 9	3
<b>2</b>	<b>Jugement majoritaire</b>	<b>3</b>
2.1	Question 10	3
2.2	Question 11	4
2.3	Question 12	4
2.4	Question 13	4
2.5	Question 14	4
2.6	Question 15	4
2.7	Question 16	4
2.8	Question 17	4
2.9	Question 18	5

## 1 Scrutin uninominal

### 1.1 Question 1

```
type candidat = string;; (* Réstriction aux chaînes représentant un candidat. *)
type bulletin = candidat;;
type urne = bulletin list;;
type score = int;; (* Réstriction aux entiers positifs. *)
type panel = candidat list;;
```

### 1.2 Question 2

#### Spécification

Sémantique	compte c u renvoie le score d'un candidat c pour une urne u.
Profile	compte: candidat -> urne -> score
Exemple	<pre>compte "Eric" ["Eric"; "Eric"; "Mathias"] = 2 compte "Mathias" ["Eric"; "Eric"] = 0</pre>

## Implémentation

```
let rec compte (c: candidat) (u: urne) : score =  
  match u with  
  | [] -> 0  
  | t::q -> if t=c then 1+(compte c q) else (compte c q);;
```

## 1.3 Question 3

### Spécification

Sémantique	depouiller lc u renvoie la liste des scores par candidat à partir d'un panel lc et d'une urne u.
Profile	depouiller: panel -> urne -> resultat list
Exemple	<pre>depouiller ["Eric"; "Mathias"] ["Eric"; "Eric"; "Mathias"] = [("Eric", 2);   ↪ ("Mathias", 1)] depouiller ["Eric"; "Mathias"] ["Eric"; "Eric"] = [("Eric", 2); ("Mathias",   ↪ 0)]</pre>

## Implémentation

```
type resultat = candidat*score;;  
let rec depouiller (lc: panel) (u: urne) : resultat list =  
  match lc with  
  | [] -> []  
  | t::q -> (t, compte t u)::(depouiller q u);;
```

## 1.4 Question 4

### Spécification

Sémantique	union r1 r2 renvoie l'union de deux résultats r1 et r2 pour un même candidat. La fonction renvoie une erreur si les candidats sont différents.
Profile	union: resultat -> resultat -> resultat
Exemple	<pre>union ("Eric", 2) ("Eric", 7) = ("Eric", 9) union ("Eric", 7) ("Mathias", 2) = "Les candidats sont différents."</pre>

## Implémentation

```
let union (r1: resultat) (r2: resultat) : resultat =  
  let (c1, s1)=r1 and (c2, s2)=r2 in  
  if c1=c2 then (c1, s1+s2) else failwith "Les candidats sont différents.";;
```

## 1.5 Question 5

### Spécification

Sémantique	max_depouiller l renvoie le résultat du candidat qui possède le meilleur score parmi les éléments présents dans la liste de résultat l. La fonction renvoie une erreur si la liste est vide.
Profile	max_depouiller: resultat list -> resultat
Exemple	<pre>max_depouiller [("Eric", 7); ("Mathias", 2)] = ("Eric", 7) max_depouiller [] = "Il n'y a pas de résultat."</pre>

## Implémentation

```
let rec max_depouiller (l: resultat list) : resultat =
  match l with
  | [] -> failwith "Il n'y à pas de résultat."
  | [e] -> e
  | t::q -> let (c1, s1)=t and (c2, s2)=(max_depouiller q) in
    if s1>s2 then t else (c2, s2);;
```

## 1.6 Question 6

### Spécification

Sémantique	vainqueur_scrutin_uninominal u lc renvoie le candidat qui possède le meilleur score parmi les bulletins d'une urne u et les candidats d'un panel lc. La fonction renvoie une erreur si le panel est vide.
Profile	vainqueur_scrutin_uninominal: urne -> panel -> candidat
Exemple	<pre>vainqueur_scrutin_uninominal ["Eric"; "Eric"; "Mathias"] ["Eric"; "Mathias"] ↪ = "Eric" vainqueur_scrutin_uninominal ["Eric"; "Eric"; "Mathias"] [] = "Il n'y à pas ↪ de résultat."</pre>

## Implémentation

```
let vainqueur_scrutin_uninominal (u: urne) (lc: panel) : candidat =
  let (c, s)=max_depouiller (depouiller lc u) in c;;
```

## 1.7 Question 7

### Spécification suppr\_elem

Sémantique	suppr_elem l e renvoie la liste l sans la première occurrence de l'élément e.
Profile	suppr_elem: 'a list -> 'a -> 'a list
Exemple	<pre>suppr_elem ["Eric"; "Mathias"] "Mathias" = ["Eric"]</pre>

## Implémentation suppr\_elem

```
let rec suppr_elem (l: 'a list) (e: 'a) : 'a list =
  match l with
  | [] -> []
  | t::q -> if t=e then q else t::(suppr_elem q e);;
```

### Spécification deux\_premiers

Sémantique	deux_premiers u lc renvoie le couple de résultats des deux candidats qui possèdent le meilleur score parmi les bulletins d'une urne u et les candidats d'un panel lc. La fonction renvoie une erreur si le panel est vide.
Profile	deux_premiers: urne -> panel -> resultat*resultat
Exemple	<pre>deux_premiers ["Eric"; "Eric"; "Mathias"] ["Eric"; "Mathias"] = (("Eric", ↪ 2), ("Mathias", 1)) deux_premiers ["Eric"; "Eric"; "Mathias"] [] = "Il n'y à pas de résultat."</pre>

## Implémentation deux\_premiers

```
let deux_premiers (u: urne) (lc: panel) : resultat*resultat =  
  let (c1, s1)=(max_depouiller (depouiller lc u)) in  
  let c2=(max_depouiller (depouiller (suppr_elem lc c1) u)) in  
  ((c1, s1), c2);;
```

### 1.8 Question 8

à remplir

### 1.9 Question 9

à remplir

## 2 Jugement majoritaire

### 2.1 Question 10

Dans le cadre d'une élection avec 12 candidats il y a  $6^{12} = 2176782336$  bulletins possibles que l'on peut mettre dans l'urne.

à compléter

### 2.2 Question 11

```
type mention = Arejeter | Insuffisant | Passable | Assezbien | Bien | Tresbien;;  
type bulletin_jm = mention list;;  
type urne_jm = bulletin_jm list;;
```

### 2.3 Question 12

```
let rec depouille_jm (u: urne_jm) : mention list list =  
  match u with  
  | [] -> []  
  | e -> (List.map (List.hd) u)::(  
    match (List.hd u) with  
    | [] -> []  
    | [e] -> []  
    | e -> depouille_jm (List.map (List.tl) u)  
  );;
```

### 2.4 Question 13

```
let tri (l:'a list):'a list = List.sort compare l;;  
let tri_mentions (u: mention list list) : mention list list = List.map (tri) u;;
```

### 2.5 Question 14

```
let mediane (l: 'a list) : 'a = List.nth 1 ((List.length l) / 2);;
```

### 2.6 Question 15

```
let meilleur_mediane (u: mention list list) : mention =  
  let medList = List.map (mediane) (tri_mentions u) in  
  tri medList |> List.rev |> List.hd;;
```

## 2.7 Question 16

```
let supprime_perdants (u: mention list list) : mention list list =  
  let med = meilleur_mediane u in  
  List.map (  
    fun (l: mention list) : mention list -> if mediane l < med then [] else l  
  ) u;;
```

## 2.8 Question 17

```
let rec supprime_mention (l: mention list) (e: mention) : mention list =  
  match l with  
  | [] -> []  
  | t::q -> if t = e then q else t::(supprime_mention q e);;  
  
let supprime_meilleure_mediane (u: mention list list) : mention list list =  
  List.map (  
    fun (l: mention list) : mention list ->  
      if l = [] then [] else l |> mediane |> supprime_mention l  
  ) u;;
```

## 2.9 Question 18

content...