

Van-Lu Nguyen

Partiel MPSOC - Draft

Novembre 2004

Les exercices sont indépendants.

Il est souvent utile, pour répondre à une question, d'utiliser les fonctions demandées dans les questions précédentes.

La clarté des réponses et la présentation des programmes seront appréciées.

Le barème (total sur 24) est donné à titre indicatif.

Exercice(s)

Exercice 1 – Jeu de Scrabble

(14 points)

Le scrabble est un jeu qui se joue avec des pions où sont écrits des lettres et leur nombre de points associés. Un pion est donc caractérisé par une lettre et une valeur associée à la lettre. Le but du jeu est de construire des mots totalisant le plus grand nombre de points. Le tableau ci-dessous indique les caractéristiques d'un pion, c'est-à-dire le nombre de points associés à chaque lettre (* représente un joker).

*	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	3	3	2	1	4	2	4	1	8	10	1	2	1	1	3	8	1	1	1	1	4	10	10	10	10

Question 1

Construire en Ocaml le type d'enregistrement `pion` ayant deux champs : `lettre` et `valeur`, qui permet la déclaration des pions ayant des caractéristiques ci-dessus indiquées.

Solution:

```
type pion = {lettre: char; valeur: int};;
```

Question 2

Ecrire une fonction `fabrique_pion` qui prend une lettre en argument et qui retourne un pion correspondant à cette lettre. Rappelons qu'en Ocaml, il est possible d'abrégier les motifs contigus par un motif intervalle comme le montre l'exemple ci-dessous.

```

let kesako c=
match c with
| '0'..'9' -> ``digit``
| 'a'..'z' -> ``minuscule``
| 'A'..'Z' -> ``Majuscule``
| _ -> ``divers``
;;

```

Solution:

```

let fabrique_pion c =
  match c with
  | '*' -> {lettre=c;valeur=0}
  | 'A'|'E'|'I'|'L'|'N'..'O'|'R'..'U' -> {lettre=c;valeur=1}
  | 'D'|'G'|'M' -> {lettre=c;valeur=2}
  | 'B'|'C'|'P' -> {lettre=c;valeur=3}
  | 'F'|'H'|'V' -> {lettre=c;valeur=4}
  | 'J'|'Q' -> {lettre=c;valeur=8}
  | 'K'|'W'..'Z' -> {lettre=c;valeur=10}
  | _ -> raise (Invalid_argument ``val``)
;;

```

Question 3

Ecrire une fonction `comptage_points` qui prend en argument un mot – mot étant une liste non vide de pions – et qui retourne le nombre de points correspondant à ce mot.

Solution:

```

let comptage_points lp =
  match lp with
  | [] -> 0
  | t::q -> (t.valeur)+(comptage_points q)
;;

```

Le tableau ci-dessous indique le nombre de pions pour chaque lettre.

*	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
2	9	2	2	3	15	2	2	2	8	1	1	5	3	6	6	2	1	6	6	6	6	2	1	1	1	1

Question 4

Ecrire une fonction `occurrences` qui prend une lettre en argument et qui retourne le nombre de pions correspondant à cette lettre.

Solution:

```
let occurrences c =
  match c with
  | 'J'..'K' | 'Q' | 'W'..'Z' -> 1
  | 'B'..'C' | 'F'..'H' | 'P' | 'V' -> 2
  | 'D' | 'M' -> 3
  | 'L' -> 5
  | 'N'..'O' | 'R'..'U' -> 6
  | 'I' -> 8
  | 'A' -> 9
  | 'E' -> 15
  | '*' -> 2
  | _ -> raise (Invalid_argument ``occurrences``)
```

Question 5

Ecrire une fonction `total ()` qui, grâce à la fonction `occurrences`, calcule le nombre total des pions dans le jeu.

Solution:

```
let total () =
  let tox=ref 2 in
  for i=0 to 25 do
    let c=char_of_int (i+int_of_char('A')) in
    let ox=occurrences c in
    tox:=!tox+occurrences c;
  done;
  !tox;;
```

On se propose le type d'enregistrement `sac` qui permet de représenter le jeu de Scrabble.

```
type sac={mutable cardinal: int;les_pions: pion array};;
```

Le champ `cardinal` indique le nombre de pions non encore tirés, et `les_pions` représentant l'ensemble de tous les pions, chaque pion étant représenté dans une case distincte (par exemple, deux pions 'A' sont dans deux cases différentes du vecteur).

Question 6

Définir une fonction `init_sac ()` qui initialise le sac de type `un_sac`.

Solution:

```

let init_sac () =
  let sak=ref [|{lettre='*';valeur=0};{lettre='*';valeur=0}|] in
  for i=0 to 25 do
    let c=char_of_int (i+int_of_char('A')) in
    let ox=occurrences c in
    for k=0 to ox-1 do
      sak:=Array.append !sak [|fabrique_pion c|];
    done;
  done;
{cardinal=total();les_pions=!sak};;

```

Le tirage des pions se fait aléatoirement parmi l'ensemble des pions non encore tirés. Pour représenter ce tirage aléatoire, il est nécessaire d'utiliser la fonction `Random.int1` retournant l'indice d'un pion de l'ensemble des pions non encore tirés.

Question 7

Définir une fonction `tirage_un_pion (sak :sac)` qui modifie le contenu du sac des pions non encore tirés lorsque qu'un pion est tiré, et retourne le pion tiré. La modification du sac se fait de la façon suivante : avant le tirage la tranche du vecteur `sak.les_pions.(i)`, $i \in [0..sak.cardinal-1]$ (resp $i \in [sak.cardinal..102]$) contient les pions non encore tirés (resp. pions déjà tirés); après le tirage d'un nombre aléatoire k , $k \in [0..sak.cardinal-1]$, les pions d'indices k et `sak.cardinal-1` seront permutés.

Solution:

```

let tirage_un_pion sak =
  let lg=sak.cardinal in
  match lg with
  | 0 -> raise (Invalid_argument ``tirage_un_pion``)
  | _ -> let nx=Random.int(lg) in
    let pion=sak.les_pions.(nx) in
      sak.les_pions.(nx)<-sak.les_pions.(sak.cardinal-1);
      sak.les_pions.(sak.cardinal-1)<-pion;
      sak.cardinal <- sak.cardinal-1;
  pion
;;

```

Exercice 2 – Permutations

(10 points)

Question 1

Définir une fonction `tissage x l` qui prend un élément x et une liste l , et qui retourne une liste des listes. Chacune de ces listes s'obtient par une insertion de x . Par exemple,

```
tissage 8 [1;2;3] ==> [[8;1;2;3];[1;8;2;3];[1;2;8;3];[1;2;3;8]]
```

```

tissage ``Il`` [``pleut``;``sur``;``Nantes``] ==>
[[``Il``;``pleut``;``sur``;``Nantes``];[``pleut``;``Il``;``sur``;``Nantes``];
 [``pleut``;``sur``;``Il``;``Nantes``];[``pleut``;``sur``;``Nantes``;``il``]]

```

¹Rappelons que la fonction `(Random.int n)` retourne une valeur entière comprise entre 0 et $n-1$

La fonction est donc de type :

```
val tissage: 'a -> 'a list -> 'a list list
```

Solution:

```
let rec tissage x l =
  match l with
  | [] -> [[]]
  | t::q -> (x::l)::(List.map (function y -> t::y) (tissage x q))
;;
```

Question 2

Définir une fonction aplatissement de type :

```
val aplatissement: 'a list list -> 'a list
```

qui renvoie la liste résultant de l'aplatissement de l comme indiquent les exemples qui suivent :

```
aplatissement [[ [1;2;3]; [2;1;3]; [2;3;1] ]; [ [1;3;2]; [3;1;2]; [3;2;1] ] ] ==>
[1;2;3]; [2;1;3]; [2;3;1]; [1;3;2]; [3;1;2]; [3;2;1]
```

```
aplatissement [
  [ '\un'';'one'';'ein'' ]; [ '\one'';'un'';'ein'' ]; [ '\one'';'ein'';'un'' ] ];
  [ '\un'';'ein'';'one'' ]; [ '\ein'';'un'';'one'' ]; [ '\ein'';'one'';'un'' ] ]
==>
[ '\un'';'one'';'ein'' ]; [ '\one'';'un'';'ein'' ]; [ '\one'';'ein'';'un'' ];
  [ '\un'';'ein'';'one'' ]; [ '\ein'';'un'';'one'' ]; [ '\ein'';'one'';'un'' ] ]
```

Solution:

```
let rec aplatissement l =
  match l with
  | [] -> []
  | t::q -> List.append t (aplatissement q)
;;
```

Question 3

Définir une fonction permutations l qui renvoie la liste de toutes les permutations de la liste l.

Solution:

```
let rec permutations l =
  match l with
  | [] -> [l]
  | t::q -> aplatissement (List.map (tissage t) (permutations q))
;;
```