TP n9 – Types construits en OCaml

1. Type enregistrement

Dans cette partie on va manipuler des complexes sous forme d'un type enregistrement.

- **Q1.** Définir un type enregistrement `complexe` pour représenter les complexes avec une partie réelle et une partie imaginaire flottante.
- **Q2.** Écrire l'addition `addition : complexe -> complexe -> complexe` et la soustraction `soustraction : complexe -> complexe -> complexe` pour les complexes.
- **Q3.** Écrire une fonction pour calculer le conjugué d'un complexe. `conjugue : complexe -> complexe`
- **Q4.** Écrire la multiplication des complexes. `multiplication : complexe -> complexe -> complexe`
- **Q5.** Écrire une fonction qui calcule le module d'un complexe. `modu : complexe -> float`
- **Q6.** **Bonus** Définir un autre type pour la forme polaire (ou exponentielle ou trigonométrique) d'un complexe et écrire deux coercitions (fonctions `algebrique_of_polaire` et `polaire_of_algebrique`) pour passer d'un type à l'autre.

2. Type énumération avec paramètres

On définit la couleur d'une carte par le type suivant. Le type `carte` prend en argument la couleur de la carte :

```
type couleur =
| Pique
| Coeur
| Carreau
| Trefle
;;
```

```
type carte =
| Joker
| As of couleur
| Point of int*couleur
...
;;
```

- **Q7.** Compléter la définition du type `carte` pour représenter toutes les cartes possibles.
- **Q8.** Définir le roi de cœur, la dame de pique et le neuf de trèfle.

Rappel sur le filtrage :

En Ocaml on peut faire une disjonction de cas sur la forme des entrées (les constructeurs utilisés).

Par exemple si on veut écrire une fonction qui renvoit vrai si et seulement si la carte en entrée est un as :

```
let est_as c = match c with
|As _ -> true (*Il faut que ce soit un as, mais la couleur n'est pas importante*)
|_ -> false (*Pour toute autre chose, on renvoie faux*)

(*D'autres exemples de filtres, ici inutiles*)
|As Carreau -> true (*la carte est un as de carreau*)
|Point (x,Pique) -> false ;; (*c'est une carte valeur x, sa valeur est x et c'est un pique*)
```

- **Q9.** Voici le début d'une fonction déterminant si une carte est un carreau. Complétez la. Quel est son type ?

```
let est_carreau carte = match carte with
| Joker -> false
| As Carreau -> true
...
;;
```

- **Q10.** Prévoir et expliquer ce que renvoient les expressions suivantes :
  - `let carte1 = Joker;;`
  - `Pique;;`
  - `carte1 = carte2;;`
  - `As;;`
  - `(As Carreau, Roi Pique, Valet Trefle);;`
- **Q11.** Écrire une fonction `est_figure : carte -> bool` qui indique si la carte reçue en entrée est une figure ou non. Une figure est un roi, un valet ou un reine.

3. Types récursifs

On définit un type récursif coloration pour définir les couleurs par synthèse soustractive :

```
type coloration =
  | Cyan
  | Magenta
  | Jaune
  | Melange of coloration * coloration
;;
```

- **Q12.** Définir en Caml le rouge (mélange de magenta et de jaune) puis l'orange.
- **Q13.** Que fait la fonction suivante ?

```
let rec cmj color =
  match color with
  | Cyan -> (1., 0., 0.)
  | Magenta -> (0., 1., 0.)
  | Jaune -> (0., 0., 1.)
  | Melange (color1, color2) ->
     let c1, m1, j1 = cmj color1 in
     let c2, m2, j2 = cmj color2 in
     ((c1 +. c2) /. 2.,
     (m1 +. m2) /. 2.,
     (j1 +. j2) /. 2.)
;;
```

4. Pierre-feuille-ciseau

Dans cette partie, on veut écrire une fonction capable de simuler une partie de Pierre-Feuille-Ciseau (où l'ordinateur joue contre l'ordinateur).

Pour tirer les coups au hasard, on va utiliser le module `Random`. Il suffit de placer la ligne `Random.self_init();;` une seule fois dans le fichier, puis, à chaque fois qu'on veut tirer une valeur, on fait un appel à `Random.int b` pour tirer un entier au hasard entre 0 et b (b exclu). Le tirage aléatoire peut être fait dans une fonction.

Par exemple, `Random.int 3` tire un nombre valant soit 0, soit 1, soit 2, avec une probabilité uniforme.

- **Q14.** Définir un type énumération `coup` qui décrit les trois coups possibles du jeu.
- **Q15.** Définir un type énumération `resultat` qui représente une victoire, une défaite, ou une égalité.
- **Q16.** Définir un alias `manche` pour un couple de coups. Un objet de type `manche` représente le coup joué par les deux joueurs au cours d'une manche.
- **Q17.** Définir une fonction `tour : unit -> manche` qui renvoie une manche aléatoire. Vous pourrez utiliser la fonction `Random.int`.
- **Q18.** Écrire une fonction `resultat_tour : manche -> resultat` qui étant donné une manche, donne le résultat du point de vue du premier joueur.
- **Q19.** Définir un type enregistrement `score` à deux champs représentant respectivement le score du premier joueur et celui de son adversaire.
- **Q20.** Écrire une fonction `maj_score : score -> resultat -> score` qui renvoie un score mis à jour en fonction du résultat d'une manche.
- **Q21.** Écrire une fonction `resultat_jeu : score -> resultat` qui étant donné le score actuel d'une partie, donne le résultat du point de vue du premier joueur.
- **Q22.** Écrire une fonction `jeu : int -> resultat` qui simule une partie dont le nombre de manches est passé en paramètre.
  Cette fonction doit être récursive. On pourra par exemple itérer tant qu'il reste des manches à faire.