Devoir en temps libre n° 1

Ce devoir est à préparer pour le lundi 26 février 2018 et sera à rendre au début du TP. Vous devez amener[1] votre code CAML avec vous, sous la forme d'un unique fichier `.ml` dont le nom sera obligatoirement et exactement `<login>.ml` où `<login>` est votre nom d'utilisateur du laboratoire d'informatique (tout en minuscules avec les tirets).

Votre programme devra respecter scrupuleusement les noms et les types indiqués (ou un type plus général). Le code devra compiler sans erreurs ni aucun message d'avertissement. L'encodage du fichier devra être UTF-8 et vous n'utiliserez que des caractères ASCII pour les noms de variables. Il est impératif de veiller au strict respect de ces consignes.

EXERCICE 1 *Représentation des nombres rationnels*

On représente en CAML un rationnel par un couple $(a, b) \in \mathbb{Z} \times \mathbb{Z}^*$. En CAML, on utilise donc le type `int * int`. Écrire les fonctions suivantes en CAML :

1. `est_rationnel : (int * int) -> bool` qui s'évalue à vrai si et seulement si un couple d'entiers passé en argument représente un nombre rationnel.

   ✔ Il est inutile de vérifier que les éléments sont bien des entiers, puisque l'on suppose que l'argument est un couple d'entiers dans la description de la fonction.

   ⚠ Il est probable que le type de votre fonction ne soit pas `(int * int) -> bool`, mais plutôt `('a * int) -> bool`. Comme ce type est plus général, cela convient très bien. On vérifiera bien par la suite que l'on obtient soit le type demandé, soit un type plus général, comme ici.

2. `entier_vers_rationnel : int -> (int * int)` qui permet de représenter un nombre entier sous forme de nombre rationnel.
3. `multiplication : (int * int) -> (int * int) -> (int * int)` qui multiplie deux nombres rationnels entre eux. Ici encore on suppose que les arguments représentent bien des nombres rationnels et il n'est pas nécessaire de le vérifier.
4. `addition : (int * int) -> (int * int) -> (int * int)` qui ajoute deux nombres rationnels entre eux.
5. `sont_egaux : (int * int) -> (int * int) -> bool` qui teste si deux nombres rationnels sont égaux.
6. `pgcd : int -> int -> int` qui étant donné deux entiers naturels, renvoie leur plus grand diviseur commun. On pourra remarquer que si $(a, b) \in \mathbb{N} \times \mathbb{N}^*$, $\text{pgcd}(a, 0) = a$ et $\text{pgcd}(a, b) = \text{pgcd}(b, a \mod b)$ et en déduire une fonction `pgcd` récursive qui implémente l'algorithme d'Euclide.
7. `abs : int -> int` qui calcule la valeur absolue d'un entier relatif.

1. Je vous conseille de vous l'envoyer par mail depuis chez vous *et* de l'apporter sur clef USB.

8. `fraction_irreductible : (int * int) -> (int * int)` qui renvoie la représentation sous forme de fraction irréductible d'un nombre rationnel, c'est-à-dire avec le dénominateur strictement positif et avec le numérateur et dénominateur premiers entre eux. On pourra utiliser les fonctions `pgcd` et `abs` et on fera bien attention aux signes.

EXERCICE 2

Imaginer et écrire des fonctions simples mais non triviales ayant les types suivants :

- `f1 : int -> (float -> float)`
- `f2 : (int -> float) -> float`
- `f3 : int -> float -> float`
- `f4 : (int -> float) -> (int -> float)`
- `f5 : int -> (float -> int -> float)`
- `f6 : int -> (float -> int) -> float)`
- `f7 : int -> float -> int -> float`
- `f8 : (int -> float -> int) -> float`

On cherchera la définition mathématique avant de se lancer dans l'implémentation.

EXERCICE 3 *Différences finies*

L'opérateur des différences finies Δ associe à toute suite réelle $(u_n)_{n\in\mathbb{N}}$ la suite réelle $(u_{n+1} - u_n)_{n\in\mathbb{N}}$. Écrire une fonction `delta` qui réalise cette transformation. On commencera par bien réfléchir à son type. On se souviendra qu'une suite n'est rien d'autre qu'une fonction particulière.

EXERCICE 4 *Produit*

Écrire une fonction `produit : (float -> float) -> int -> int -> float` telle que, pour $f : \mathbb{R} \to \mathbb{R}$ et $n, m \in \mathbb{N}$ quelconques, `(produit f n m)` calcule le produit :

$$\prod_{k=n}^{m} f(k)$$

En *déduire* la fonction `factorielle : int -> int`.

EXERCICE 5 *Itération*

Écrire une fonction récursive `iter : (int -> 'a -> 'a) -> 'a -> int -> 'a` telle que l'expression `iter f x n` s'évalue en la valeur de $f(n, (f(n-1, f(n-2, \ldots, f(1, x)))))$.

EXERCICE 6 *Bonus ! (Facultatif)*

Écrire une fonction `nombre_ecriture : int -> int -> int` telle que l'évaluation de `nombre_ecriture n m` renvoie le nombre d'écritures (sans tenir compte de l'ordre) de n comme somme de m entiers strictement positifs [2]. Par exemple, le nombre d'écritures de 6 comme somme de deux entiers strictement positifs est 3 car $6 = 1+5 = 2+4 = 3+3$ et il n'y a pas d'autres possibilités ; comme somme de quatre entiers strictement positifs, il y a deux possibilités.

2. Indication : On pourra chercher une relation de récurrence en séparant deux cas, selon que l'entier 1 est utilisé ou non dans la décomposition.