Programmation fonctionnelle avec OCaml
TP 1

Objectifs :
- Prise en main de l'environnement de développement : terminal avec `bash`, éditeur `emacs`, interpréteur `ocaml`, documentation.
- Programmation fonctionnelle : type algébrique, filtrage de motif, récursivité.

Pour ce premier TP, éditer le code source avec `emacs` puis l'évaluer et le tester (avec un copier-coller) grâce à « l'interpréteur » (*top-level interactive loop*) `ocaml`.

Plus grand commun diviseur (encore...)

Écrire une fonction **récursive** pour calculer le plus grand commun diviseur de deux nombres entiers en utilisant l'algorithme d'Euclide :

```
# pgcd 3822 2310;;
- : int = 42
```

Rappel : $\text{pgcd}(a, 0) = a$ *et* $\text{pgcd}(a, b) = \text{pgcd}(b, a \mod b), \forall b \neq 0$.

Sequence d'entiers

1. Définir le **type algébrique récursif** `int_seq` qui correspond soit à la séquence vide, soit à un nœud composé d'un élément entier et de la suite de la séquence. Définir la séquence $\langle 1, 2, 3 \rangle$.
2. Écrire une fonction `length: int_seq -> int` qui renvoie la taille d'une séquence.
   ```
   # length s;;
   - : int = 3
   ```
3. Écrire une fonction `sum: int_seq -> int` qui renvoie la somme des éléments d'une séquence.
   ```
   # sum s;;
   - : int = 6
   ```
4. Écrire une fonction `map: (int -> int) -> int_seq -> int_seq` qui applique une fonction à tous les éléments d'une séquence et renvoie la séquence correspondante. Tester avec une fonction anonyme qui renvoie le carré (entier de son argument).
5. Écrire une fonction `filter: (int -> bool) -> int_seq -> int_seq` qui renvoie la séquence des éléments vérifiant un prédicat (pris en premier paramètre). Tester avec une fonction anonyme qui sélectionne les entiers impairs.
6. Écrire une fonction qui combine les deux précédentes
   `comprehension: (int -> int) -> (int -> bool) -> int_seq -> int_seq`
   similaire à la construction de liste en compréhension en Python :
   `comprehension f p l` ≡ `[f(x) for x in l if p(x)]`

7. Écrire `map2: (int -> int -> int) -> int_seq -> int_seq -> int_seq` équivalente à `map` mais qui applique la fonction aux éléments de deux séquences simultanément (en terminant dès que l'une est vide).
8. Toutes les fonctions écrites de la 2e à la 6e question sont des cas particuliers d'un itérateur générique `iter: (int -> 'a -> 'a) -> int_seq -> 'a -> 'a` tel que `iter f seq z` calcule `f x1 (f x2 (... (f xn z)...))` si la séquence est $\langle x_1, x_2, \ldots, x_n \rangle$. Écrire cet itérateur.
9. Utiliser cet itérateur pour réécrire directement les fonctions `length`, `sum` et `map`.
10. Écrire la fonction `prod: int_seq -> int_seq -> int_seq` qui calcule le produit des éléments des couples correspondant au produit cartésien des deux listes prises en paramètre, à l'aide d'une double application de l'itérateur. Par exemple, l'application de `prod` aux séquences $\langle 1, 2, 3 \rangle$ et $\langle 4, 5 \rangle$ doit renvoyer la séquence $\langle 4, 5, 8, 10, 12, 15 \rangle$ correspondant à $\langle 1 \times 4, 1 \times 5, 2 \times 4, 2 \times 5, 3 \times 4, 3 \times 5 \rangle$.