TD2 – Fonctions (suite) – Listes

Exercice 2.1 (PGCD).
Le PGCD (plus grand commun diviseur) de deux entiers naturels a et b tels que $a \geq b$ peut être défini récursivement par :

$$pgcd(a,b) = \begin{cases} b & \text{si } r = 0 \\ pgcd(b,r) & \text{si } r \neq 0 \end{cases}$$

où r est le reste de la division entière de a par b.

1. Définir une fonction de signature `pgcd (n : int) (m : int) : int` qui calcule le PGCD de `n` et `m`. Cette fonction ne devra pas effectuer deux fois le même calcul du reste d'une division entière.

```
# (pgcd 96 36);;          # (pgcd 36 96);;          # (pgcd 7 19);;
- : int = 12              - : int = 12              - : int = 1
```

2. Quel est le type de la fonction `pgcd` ?

Exercice 2.2 (Quelques fonctions sur les listes).

1. Définir une fonction de signature `repeat (n : int) (e : 'a) : 'a list` qui, étant donnés un entier n et un élément e de type `'a`, construit la liste contenant n occurrences de e.

```
# (repeat 0 'p');;              # (repeat 4 'p');;
- : char list = []              - : char list = ['p'; 'p'; 'p'; 'p']
```

2. Définir une fonction de signature `all_in_list (l1 : 'a list) (l2 : 'a list) : bool` qui détermine si tous les éléments de la liste `l1` sont dans la liste `l2`.
Indication. Utiliser la fonction prédéfinie `List.mem : 'a -> 'a list -> bool` qui détermine si un élément est présent dans une liste.

```
# (all_in_list [] [2; -1]);;          # (all_in_list [-3; 8] [2; -3; -5; 8; -1]);;
- : bool = true                       - : bool = true
# (all_in_list [4; 8] []);;           # (all_in_list [-3; 4; 8] [2; -3; -5; 8; -1]);;
- : bool = false                      - : bool = false
```

3. Définir une fonction de signature `range_inter (a : int) (b : int) : int list` qui construit la liste des entiers consécutifs de l'intervalle [a,b]. Lorsque `a > b`, cette fonction retourne la liste vide.

```
# (range_inter 7  2);;  # (range_inter 2  7);;          # (range_inter 2  2);;
  - : int list = []     - : int list = [2; 3; 4; 5; 6; 7]  - : int list = [2]
```

4. Définir une fonction de signature `map_cons (e : 'a) (l : 'a list list) : 'a list list` qui étant donnés un elément `e` et une liste de listes `l` construit la liste de listes obtenue en ajoutant en tête de chaque liste de `l` l'élément `e`.

```
# (map_cons 3 []);;      # (map_cons 'x' [ [] ]);;
- : int list list = []   - : char list list = [ ['x'] ]
# (map_cons true [ [true; false; false]; [false; true]; [] ]);;
- : bool list list = [[true; true; false; false]; [true; false; true]; [true]]
```

Exercice 2.3 (Préfixes d'une liste – Extrait Examen Juin 2023).
Une liste `l1` est un préfixe d'une liste `l2` s'il existe une liste `l3` telle que `l1 @ l3 = l2`. Par exemple la liste `['b'; 'a']` est un préfixe de la liste `['b'; 'a'; 't'; 'e'; 'a'; 'u']`.

1. Définir une fonction de signature `is_prefix (l1 : 'a list) (l2 : 'a list) : bool` qui détermine si la liste `l1` est un préfixe de la liste `l2`.

```
# (is_prefix [] [1;2;3]);;
- : bool = true
# (is_prefix [1;2] [1;2]);;
- : bool = true
# (is_prefix [1;2;3] [1;2]);;
- : bool = false
# (is_prefix [1;2] [1;2;3;4;5]);;
- : bool = true
# (is_prefix [2;3] [1;2;3]);;
- : bool = false
```

2. Définir une fonction de signature `prefixes (l : 'a list): 'a list list` qui étant donnée une liste `l` construit la liste de toutes les listes correspondant à un préfixe de `l`.

```
# (prefixes []);;
- : 'a list list = [ [] ]
# (prefixes ['x']);;
- : 'a list list = [ []; ['x'] ]
# (prefixes ['h'; 'o'; 'u'; 'x']);;
- : char list list = [ []; ['h']; ['h'; 'o']; ['h'; 'o'; 'u'];
                       ['h'; 'o'; 'u'; 'x'] ]
# (prefixes ['c'; 'h'; 'o'; 'u'; 'x']);;
- : char list list = [ []; ['c']; ['c'; 'h']; ['c'; 'h'; 'o'];
                       ['c'; 'h'; 'o'; 'u']; ['c'; 'h'; 'o'; 'u'; 'x'] ]
```

Indications.
– On pourra utiliser la fonction `map_cons` définie dans l'exercice 2.2.
– La liste vide est un préfixe de toutes les listes.
– Une liste est préfixe d'elle même puisque `l @ [] = l` .
– Le résultat de `prefixes` n'est jamais une liste vide puisqu'il contient toujours la liste vide.

3. Définir une fonction de signature `all_prefix_in (l1 : 'a list) (l2 : 'a list list) : bool` qui détermine si tous les préfixes de la liste `l1` sont des éléments de la liste de listes `l2`.
Indications. Utiliser la fonction `all_in_list` définie dans l'exercice 2.2.

```
# (all_prefix_in [1;0;0] [[];[0];[0;1];[1];[1;0];[1;0;0];[1;1];[1;1;0];[1;1;1]]);;
- : bool = true
# (all_prefix_in [1;0;0] [[];[0];[0;1];[1];[1;0;0];[1;1];[1;1;0];[1;1;1]]);;
- : bool = false
```