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);;
- : int = 12
# (pgcd 36 96);;
- : int = 12
# (pgcd 7 19);;
- : 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 'a');;
- : char list = []
# (repeat 4 'a');;
- : char list = ['a'; 'a'; 'a'; 'a']
```

2. Définir une fonction de signature `abs_list (l : int list) : int list` qui construit la liste contenant les valeurs absolues des éléments de `l` (sans changer l'ordre de ses éléments).
Indication. Utiliser la fonction prédéfinie `abs : int -> int` qui calcule la valeur absolue d'un entier.

```
# (abs_list [2; -3; -5; 8; -1]);;
- : int list = [2; 3; 5; 8; 1]
```

3. Définir une fonction de signature `max_list (l : int list) : int` qui détermine le plus grand entier d'une liste non vide d'entiers. L'exécution de la fonction `max_list` avec la liste vide pour argument lèvera l'exception `Invalid_argument`.
Indication. Utiliser la fonction prédéfinie `max : 'a -> 'a -> 'a` qui lorsqu'elle est appliquée sur deux entiers retourne le plus grand de ces deux entiers.

```
# (max_list []);;
Exception: Invalid_argument "empty list".
#  (max_list [2; -3; -5; 8; -1]);;
- : int = 8
```

4. Définir une fonction de signature `rm_last (l : 'a list) : 'a list` qui construit la liste des éléments de `l` privée de son dernier élément. On suppose que `l` est une liste non vide (l'exécution de la fonction `rm_last` avec la liste vide pour argument lèvera l'exception `Invalid_argument`).

```
# (rm_last []);;
Exception: Invalid_argument "empty list".
#  (rm_last [2; -3; -5; 8; -1]);;
- : int list = [2; -3; -5; 8]
```

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

6. 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 []);;
- : int list list = []
# (map_cons 'x' [ [] ]);;
- : 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)**.**

1. Définir une fonction de signature `rm_pref (n : int) (l : 'a list): 'a list` qui construit une liste obtenue en supprimant les `n` premiers éléments de `l`. Si `n` est strictement supérieur à la longueur de la liste `l`, le résultat est la liste vide.

```
# (rm_pref 3 [1;2]);;
- : int list = []
# (rm_pref 0 [1;2]);;
- : int list = [1; 2]
# (rm_pref 3 [1;2;3;4;5]);;
- : int list = [4; 5]
```

2. Définir une fonction de signature `lg_prefix (l : 'a list) : int` qui calcule le nombre d'éléments identiques au début de la liste `l` et renvoie 0 si `l` est vide.

```
# (lg_prefix []);;
- : int = 0
# (lg_prefix [false]);;
- : int = 1
# (lg_prefix [2;1]);;
- : int = 1
# (lg_prefix ["a";"a";"a";"b";"b";"c"]);;
- : int = 3
```

3. 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']`. 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 **??**.

– 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.