TD3 – Types produits – Fonctionnelles sur les listes

Exercice 3.1 (Fréquences d'apparition des éléments d'une liste)**.**
A partir d'une liste L (dont les éléments sont de type quelconque), on souhaite construire une liste contenant pour chaque élément c de L une paire (c, n) où n est la fréquence d'apparition de c dans L.

1. Définir une fonction de signature `add_freq (c : 'a) (l : ('a*int) list) : ('a*int) list` qui étant donnés un élément c et une liste l de paires (c_i, n_i) construit la liste l :
   - dans laquelle est ajoutée la paire $(c, 1)$ si aucune paire ne correspond au symbole c dans l (i.e. la liste l ne contient aucune paire (c, n))
   - dans laquelle la paire (c_i, n_i) apparaissant dans l est remplacée par la paire $(c_i, n_i + 1)$ lorsque $c_i = c$

   Exemples :
   - `(add_freq 'A' [])` construit la liste `[('A', 1)]`
   - `(add_freq 'A' [('D',1); ('E',1); ('A',5); ('B',3); ('C',1)])` construit la liste `[('D', 1); ('E', 1); ('A', 6); ('B', 3); ('C', 1)]`
   - `(add_freq 'H' [('D',1); ('E',1); ('A',5); ('B',3); ('C',1)])` construit la liste `[('D', 1); ('E', 1); ('A', 5); ('B', 3); ('C', 1); ('H', 1)]`
2. Définir une fonction de signature `freq (l : 'a list) : ('a * int) list` qui prend en argument une liste l et qui construit la liste de toutes les paires (c, n) où c est un élément apparaissant dans l et n est sa fréquence d'apparition dans l. On pourra utiliser la fonction `add_freq` lors de cette construction.

   Exemple :
   `(freq ['A';'A';'B';'A';'C';'B';'A';'G';'H';'A';'A';'F';'E';'A';'D';'B';'A'])`
   construit la liste
   `[('A',8); ('B',3); ('D',1); ('E',1); ('F',1); ('H',1); ('G',1); ('C',1)]`

Exercice 3.2 (Listes bien balisées – Extrait Partiel Novembre 2021)**.**
Beaucoup de langages informatiques utilisent des symboles « ouvrants » et « fermants » pour structurer du texte. Par exemple, OCaml utilise les parenthèses dans les expressions et XML et HTML utilisent des *balises* (par exemple les balises notées `<p>` et `</p>` délimitent un paragraphe). Dans cet exercice, pour simplifier, on utilise les caractères `'<'` et `'>'` pour indiquer les balises (ou parenthèses) ouvrantes et fermantes. Pour simplifier encore, on considère des listes de caractères plutôt que des textes. Et pour simplifier complètement, on ne considère que des listes qui ne contiennent que les caractères `'<'` et `'>'`.

Une liste ℓ ne contenant que les caractères `'<'` (balise *ouvrante*) et `'>'` (balise *fermante*) est dite *bien balisée* si et seulement si elle vérifie les deux conditions suivantes :

1. ℓ contient exactement autant de balises ouvrantes que de balises fermantes
2. tout préfixe de ℓ contient un nombre de balises ouvrantes supérieur ou égal au nombre de balises fermantes

Par exemple :

- `['<';'<';'>';'<';'>';'>';'<';'>']` est bien balisée
- `['<';'>';'<';'>';'>';'<']` est mal balisée : la liste contient le même nombre de balises ouvrantes que de balises fermantes mais le préfixe `['<';'>';'<';'>';'>']` de cette liste contient plus de balises fermantes que de balises ouvrantes

— `['<';'>';'<';'>';'<']` est mal balisée : tout préfixe de cette liste contient bien un nombre de balises ouvrantes supérieur ou égal au nombre de balises fermantes, mais la liste ne contient pas autant de balises ouvrantes que de balises fermantes.

L'objectif de cet exercice est de définir une fonction qui teste le bon balisage d'une liste.

1. *Nombre de balises*

   (a) Définir deux fonctions dont les signatures sont

   ```
   add_fst (c : (int * int)) : (int * int)
   add_snd (c : (int * int)) : (int * int)
   ```

   telles que `(add_fst (n1, n2)) = (n1+1, n2)` et `(add_snd (n1, n2)) = (n1, n2+1)`.

   (b) Définir une fonction de signature `nb_of (l : char list) : int * int` qui calcule le couple d'entiers (n_1, n_2) tel que n_1 est le nombre de balises ouvrantes présentes dans la liste `l` et n_2 est le nombre de balises fermantes présentes dans la liste `l`. Cette fonction ne doit effectuer qu'un unique parcours de la liste `l` pour construire le couple. On pourra utiliser les fonctions `add_fst` et `add_snd` définies à la question précédente.

   ```
   # (nb_of []);;                          # (nb_of ['<';'<';'>';'<';'>';'>';'<']);;
   - : int * int = (0, 0)                  - : int * int = (4, 3)
   ```

2. *Bon balisage*

   (a) En utilisant la fonction `nb_of` de la question précédente, définir une fonction de signature

   ```
   o_sup_f (l : char list) : bool
   ```

   qui détermine si le nombre de balises ouvrantes présentes dans la liste `l` est supérieur ou égal au nombre de balises fermantes présentes dans la liste `l`.

   ```
   # (o_sup_f ['<';'<';'>';'<';'>';'>';'<']);;    # (o_sup_f ['<';'<';'>';'>']);;
   - : bool = true                                - : bool = true
   #  (o_sup_f ['<';'>';'<';'>';'>']);;
   - : bool = false
   ```

   (b) En utilisant la fonction `List.for_all`, définir une fonction de signature

   ```
   all_o_sup_f (l : (char list) list) : bool
   ```

   qui détermine si toutes les listes d'une liste de listes `l` contiennent un nombre de balises ouvrantes supérieur ou égal au nombre de balises fermantes.

   Exemples :

   — `(all_o_sup_f [])` donne `true`

   — `(all_o_sup_f [['<';'<';'>';'>']; ['<';'<';'>';'<';'>';'>';'<']])` donne `true`

   — `(all_o_sup_f [['<';'<';'>';'>']; ['<';'>';'<';'>';'>']])` donne `false`

   (c) Déduire de ce qui précède une fonction de signature `dyck (l : char list) : bool` qui détermine si une liste ne contenant que des caractères `'<'` et `'>'` est bien balisée. On supposera que l'on dispose de la fonction `prefixes` de l'exercice 2.3 du TD2.

   Exemples :

   — `(dyck ['<';'<';'>';'<';'>';'>';'<';'>'])` donne `true`

   — `(dyck ['<';'>';'<';'>';'>';'<'])` donne `false`

   — `(dyck ['<';'>';'<';'>';'<'])` donne `false`

Exercice 3.3 (Schéma d'accumulation)**.**

1. En utilisant la fonction `List.fold_left`, définir une fonction de signature :

   ```
   sum_left (l : int list) : int
   ```

   qui étant donnée une liste d'entier `l`, calcule la somme des éléments de `l`. On supposera que la somme des éléments d'une liste vide est 0.

   Exemples :

   — `(sum_left [3; 7; 2])` calcule `12`
   — `(sum_left [-5; 10; -2; 1])` calcule `4`
   — `(sum_left [])` calcule `0`

2. Donner une autre version de `sum_left`, appelée `sum_right`) qui utilise la fonction `List.fold_right`.