info421 : Programmation fonctionnelle
TD6 : exceptions, suite

Responsables : Pierre Hyvernat et Krzysztof Worytkiewicz
Laboratoire de mathématiques de l'université de Savoie
email : Pierre.Hyvernat@... / Krzysztof.Worytkiewicz@... (...@univ-savoie.fr)
http://lama.univ-savoie.fr/~hyvernat/
http://lama.univ-savoie.fr/~worytkiewicz/

Question 1. En utilisant `List.fold_left`, programmez une fonction `somme : int list -> int` et une fonction `produit : int list -> int` qui calculent respectivement la somme et le produit des éléments d'une liste.

Question 2. Idem, mais pour une fonction `produit7` qui calcule le produit dans $\mathbf{Z}/7\mathbf{Z}$ (modulo 7). (L'opérateur "`mod`" de Caml permet de calculer le modulo.)

Question 3. Testez `produit7` sur des listes aléatoires d'éléments de $\mathbf{Z}/7\mathbf{Z}$ (en utilisant `Random.int 7` pour générer un tel élément) de grande taille. Que constatez-vous ? Expliquez.

Question 4. Réécrivez `produit7-bis` avec `List.fold_left`, mais en utilisant une exception pour améliorer le calcul.

Question 5. La fonction `open_in "..."` permet d'ouvrir un fichier, et la fonction `input_line f` permet de récupérer la prochaine ligne dans le fichier `f` :

```
let f = open_in "fichier.txt" in
let l1 = input_line f in
let l2 = input_line f in
...
```

Lorsqu'il n'y a plus de ligne dans le fichier, la fonction `input_line` lève l'exception `End_of_file`.

Écrivez une fonction qui ouvre puis récupère toutes les lignes d'un fichier dans une liste.

Question 6. Écrivez une fonction qui supprime la première occurrence d'un élément donné dans un arbre binaire de recherche : `suppr : 'a -> 'a arbre -> 'a arbre`.

Lorsque l'élément donné n'apparait pas dans l'arbre, l'utilisateur ne le saura pas, sauf s'il a utilisé une fonction `apparait` avant :

```
...
if apparait e t
then suppr e t
else l
...
```

Réécrivez `suppr : 'a -> 'a arbre -> 'a arbre` en utilisant une exception pour que l'utilisateur puisse savoir si oui ou non, l'élément qu'il tentait de supprimer était présent dans l'arbre.

Programmez une fonction `suppr_bis : 'a -> 'a arbre -> 'a arbre` à partir du nouveau `suppr`. Cette fonction aura le même comportement que l'ancienne version de `suppr`.

Question 7. La fonction `suppr_bis` de la question précédente présente un autre avantage lorsque l'élément que l'on supprime n'apparait pas dans l'arbre.

Quel est cet avantage ?

Question 8. Tartempion a écrit

```
let tete l = match l with
    [] -> raise Not_found
  | a::_ -> a

let queue l = match l with
    [] -> []
  | a::l -> l

let rec map_exn f l =
  try
    let a, l = tete l, queue l in
      (f a)::map_exn f l
  with
    Not_found -> []
```

Que fait cette fonction :
- lorsque `f` ne lève pas d'exception ?
- lorsque `f` lève une exception ?