TP 3 d'option informatique - MPSI - 19/02/2021

Ce TP se déroule sur deux séances : le 19/02/2021 et le 12/03/2021.

On rappelle qu'une liste est soit la liste vide `[]`, soit une liste non vide `(e::q)`. Dans le deuxième cas, l'élément `e` est appelé la tête de la liste et `q` est une liste appelée la queue de la liste. En général, lorsqu'on écrit une fonction `f` prenant en entrée une liste `li`, on fait donc un filtrage sur la liste :

```
let rec f li = match li with
  | []   -> ...
  | e::q -> ...
```

Exercice 1. Fonctions élémentaires sur les listes

Le but de cet exercice est de réécrire les fonctions `List.hd`, `List.tl`, `List.length`, `List.nth`, `List.rev` et `List.append` d'Ocaml.

1. Écrire une fonction `(hd: 'a list -> 'a)` qui renvoie le premier élément de la liste.
2. Écrire une fonction `(tl: 'a list -> 'a list)` qui renvoie la queue de la liste.
3. Écrire une fonction `(length: 'a list -> int)` qui renvoie la taille de la liste.
4. Écrire une fonction `(nth: 'a list -> int -> 'a)` qui renvoie l'élément d'indice n de la liste.
5. Écrire une fonction `(rev: 'a list -> 'a list)` de complexité linéaire qui renvoie la liste renversée. On pourra par exemple utiliser une fonction intermédiaire de type `('a list -> 'a list -> 'a list)`.
6. Écrire une fonction `(append: 'a list -> 'a list -> 'a list)` qui concatène deux listes.

Exercice 2. Fonctions sur les tableaux

1. Écrire une fonction `(maximum: 'a array -> 'a)` qui calcule l'élément maximum d'un tableau.
2. Écrire une fonction `(test: int array -> bool)` de **complexité linéaire** qui prend en entrée un tableau `tab` et renvoie `true` lorsque tous les éléments de $[\![0, n-1]\!]$ apparaissent dans `tab` où n est la taille de `tab`. Dans le cas contraire, votre fonction renverra `false`. Vérifiez en particulier que `(test [|1; 2|])` vaut `false`.

On souhaite réécrire la fonction `Array.to_list`.

3. Écrire une fonction `(list_of_array: 'a array -> 'a list)` qui convertit un tableau en une liste.
4. (Facultatif) Refaire la question précédente avec une fonction itérative (si votre fonction était récursive) ou récursive (si votre fonction était itérative).

On souhaite réécrire la fonction `Array.of_list`.

5. Écrire une fonction `(array_of_list: 'a list -> 'a array)` qui convertit une liste en un tableau.
6. (Facultatif) Refaire la question précédente avec une fonction itérative (si votre fonction était récursive) ou récursive (si votre fonction était itérative).

Exercice 3. Fonctions sur les listes

1. Écrire une fonction `double` qui prend en entrée une liste `li` et renvoie la liste où chaque élément de `li` apparaît deux fois consécutivement. Par exemple :

   `(double [1;2;3;4])` vaut `[1; 1; 2; 2; 3; 3; 4; 4]`.

2. Écrire une fonction `(dernier: 'a list -> 'a)` qui renvoie le dernier élément de la liste en entrée.
3. Écrire une fonction `(avant_dernier: 'a list -> 'a)` qui renvoie l'avant-dernier élément de la liste en entrée.
4. Écrire une fonction de type `('a list -> 'a -> 'a list)` qui supprime toutes les occurrences d'un élément `x` dans une liste `li`.
5. Écrire une fonction de type `(int -> 'a list -> 'a list)` qui supprime l'élément d'indice n d'une liste.
6. Écrire une fonction de type `(int -> 'a -> 'a list -> 'a list)` qui insère un élément dans une liste à l'indice n.

Exercice 4. Fonctions sur les chaînes de caractères

1. Écrire une fonction `(est_voyelle: char -> bool)` qui renvoie `true` lorsque son argument est une voyelle et `false` sinon.
2. Écrire une fonction `(nb_voyelles: string -> int)` qui compte le nombre de voyelles dans la chaîne de caractères passée en argument.
3. Écrire une fonction `est_palindrome` qui teste si une chaîne de caractères est un palindrome, c'est à dire un mot qui ne change pas lorsqu'on l'écrit à l'envers. Par exemple :

   `(est_palindrome "az")` vaut `false`, `(est_palindrome "aza")` vaut `true`,
   `(est_palindrome "azza")` vaut `true`, `(est_palindrome "azra")` vaut `false`.

Exercice 5. Prédicat sur une liste

Les fonctions demandées dans cet exercice prennent en argument une liste `(li: 'a list)` ainsi qu'une fonction `(predicat: 'a -> bool)`.

1. Écrire une fonction qui prend en entrée `li`, `predicat` ainsi qu'un entier n et renvoie le $n^{\text{éme}}$ élément de `li` qui vérifie `predicat`. Testez votre fonction.
2. Écrire une fonction `keep_true` qui renvoie la liste composée des éléments de `li` qui satisfont `predicat`.
3. Écrire une fonction `prefixe` qui renvoie le plus grand préfixe de `li` dont tous les éléments satisfont `predicat`.
4. Écrire une fonction `suffixe` qui renvoie le plus grand suffixe de `li` dont le premier élément de ne satisfait pas `predicat`.
5. Sans utiliser les questions précédentes, écrire une fonction qui renvoie le dernier élément de `li` qui vérifie `predicat`.

Exercice 6. Crible d'Eratosthène

1. Écrire une fonction qui prend en entrée un entier n et utilise le crible d'Ératostène pour déterminer le nombre de nombres premiers inférieurs ou égaux à n.
2. (Facultatif) Refaire la question précédente en utilisant un tableau (si vous avez utilisé une liste) ou une liste (si vous avez utilisé un tableau).

Exercice 7. Doublons dans une liste

1. Écrire une fonction `(tous_uniques: 'a list -> bool)` qui renvoie `true` lorsque la liste d'entrée ne contient pas d'élément en double.
2. Écrire une fonction qui supprime les doublons d'une liste en ne gardant que la dernière occurrence des doublons.
3. Écrire une fonction qui supprime les doublons d'une liste en ne gardant que la première occurrence des doublons.

Exercice 8. Préfixes d'une liste

Écrire une fonction `(liste_prefixes: 'a list -> 'a list list)` qui renvoie la liste des préfixes de la liste en argument. Par exemple :

`(liste_prefixes [])` vaut `[[]]`,
`(liste_prefixes [3; 2; 8])` vaut `[[]; [3]; [3;2]; [3;2;8]]`.

Exercice 9. Motif dans un vecteur ou dans une liste

Soient `t1` et `t2` deux tableaux et `n` la taille de `t1`. On dit que `t2` est un facteur de `t1` s'il existe $\texttt{i},\texttt{j} \in [\![\texttt{0},\texttt{n-1}]\!]$ tels que `t2` vaut `[|t1.(i); t1.(i+1); ...; t1.(j)|]`. Par exemple, les facteurs de `[|1; 2; 3|]` sont `[||]`, `[|1|]`, `[|2|]`, `[|3|]`, `[|1; 2|]`, `[|2; 3|]` et `[|1; 2; 3|]`.

1. Écrire une fonction de type `('a array -> 'a array -> bool)` qui renvoie `true` lorsque `t2` est un facteur de `t1` et `false` sinon.
2. Même question avec des listes.

On dit que `t2` est un sous-tableau de `t1` s'il existe $\texttt{k} \in \mathbb{N}$ et $\texttt{0} \leqslant \texttt{i}_1 < \texttt{i}_2 \ldots < \texttt{i}_\texttt{k} \leqslant \texttt{n-1}$ tels que `t2` vaut $\texttt{[|t1.(i}_1\texttt{), t1.(i}_2\texttt{), ..., t1.(i}_\texttt{k}\texttt{)|]}$.

3. Écrire une fonction de type `('a array -> 'a array -> bool)` qui renvoie `true` lorsque `t2` est un sous-tableau de `t1` et `false` sinon.
4. Même question avec des listes.

Exercices à rendre au plus tard le 10/03/2021

Rappel : les fonctions doivent être testées et vos tests doivent apparaître dans le fichier que vous rendez. Un seul test n'est pas suffisant et vos tests doivent être pertinents. Si vous ne respectez pas ces consignes, vous perdez des points.

Exercice 10. Fonctions sur les listes d'entiers

1. Écrire une fonction `(max: int list -> int)` qui renvoie le maximum des éléments de la liste donnée en entrée.
2. Écrire une fonction `(prod: int list -> int)` qui renvoie le produit des éléments de la liste donnée en entrée.
3. Écrire une fonction `(add: int list -> int list -> int list)` qui additionne terme à terme les éléments des deux listes en entrée. Dans le cas où les deux listes n'ont pas la même longueur, on s'arrêtera à la fin de la liste la plus courte. Par exemple :

   `(add [] [4; 2; 8])` vaut `[]`,
   `(add [] [])` vaut `[]`,
   `(add [2; 6; 9] [])` vaut `[]`,
   `(add [1; 4; 7; 2] [-2; 5; -2; 5])` vaut `[-1; 9; 5; 7]`,
   `(add [1; 4] [-2; 5; -2; 5])` vaut `[-1; 9]`,
   `(add [1; 4; 7; 2] [-2; 5])` vaut `[-1; 9]`.

4. Écrire une fonction qui prend en entrée une liste d'entiers $[a_1; ...; a_n]$ et renvoie $\prod_{\substack{i,j \in [\![1;n]\!] \\ i<j}} (a_i - a_j)$.

Exercice 11. Le compte est bon (exercice facultatif)

Dans le jeu du compte est bon, on dispose d'une liste `li` composée de 6 entiers (non nécessairement distincts) appartenant à $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$. On peut combiner ces entiers à l'aide des quatre opérations élémentaires `+`, `-`, `*` et `/` en suivant les règles suivantes :

- Chaque entier de `li` peut être utilisé au plus une fois.
- Les nombres manipulés ne peuvent pas être négatifs ou nuls.
- L'opération `/` ne peut pas être utilisée si la division ne tombe pas juste.

L'objectif est d'obtenir un résultat aussi proche que possible d'un entier noté `but`. Écrire une fonction qui prend en entrée `li` ainsi que `but`, et qui affiche une solution au jeu du compte est bon. Par exemple, avec `li = [2; 6; 75; 50; 10; 2]` et `but = 967`, votre fonction pourra afficher :

```
Resultat le plus proche: 966
Calculs:
75 - 6 = 69
2 + 2 = 4
4 + 10 = 14
14 * 69 = 966
```

Faites d'autres tests en utilisant un solveur en ligne pour vérifier que votre fonction trouve bien le résultat le plus proche (par exemple `https://www.dcode.fr/compte-est-bon`).

Exercices à rendre au plus tard le 17/03/2021

Exercice 12. Permutation dans une liste

1. Écrire une fonction `perm_droite` qui effectue une permutation circulaire à droite sur une liste. Par exemple :

   `(perm_droite [1; 2; 3; 4; 5; 6])` vaut `[6; 1; 2; 3; 4; 5]`.

2. Écrire une fonction `perm_gauche` qui effectue une permutation circulaire à gauche sur une liste. Par exemple :

   `(perm_gauche [1; 2; 3; 4; 5; 6])` vaut `[2; 3; 4; 5; 6; 1]`.

Exercice 13. Ensemble d'entiers

Dans cet exercice, on représente un ensemble d'entiers par une liste triée d'entiers sans élément en double. En particulier, les fonctions demandées ci-dessous prennent en entrée des listes triées sans doublon et doivent renvoyer des listes triées sans doublon.

1. Écrire une fonction `(union: int list -> int list -> int list)` qui renvoie l'union des deux ensembles donnés en argument.
2. Écrire une fonction `(intersection: int list -> int list -> int list)` qui renvoie l'intersection des deux ensembles donnés en argument.
3. Écrire une fonction `(difference: int list -> int list -> int list)` qui renvoie la différence des deux ensembles données en argument.
4. Écrire une fonction `(delta: int list -> int list -> int list)` qui renvoie la différence symétrique des deux ensembles donnés en argument. On rappelle que la différence symétrique de deux ensembles A et B est notée $A\Delta B$ et est définie par :

$$x \in A\Delta B \Leftrightarrow \Big[(x \in A \text{ et } x \notin B) \text{ ou } (x \notin A \text{ et } x \in B)\Big]$$

Exercice 14. Fonctionnelles itératives (exercice facultatif)

1. Écrire une fonction `(map: ('a -> 'b) -> 'a list -> 'b list)` qui prend en argument une fonction `f` ainsi qu'une liste `[a0; a1; ...; an]`, et renvoie la liste `[f a0; f a1; ...; f an]`. En d'autres termes, `map` doit avoir le même comportement que la fonction `List.map`.
2. Écrire une fonction `iter` qui prend en argument une fonction `(f: 'a -> unit)` ainsi qu'une liste `(li: 'a list)`, et applique successivement `f` à chaque élément de `li`. En d'autres termes, `iter` doit avoir le même comportement que la fonction `List.iter`. Utiliser votre fonction pour afficher tous les éléments d'une liste d'entiers.
3. Écrire une fonction `fold_left` qui prend en argument une fonction `(f: 'a -> 'b -> 'a)` ainsi qu'un argument `(a: 'a)` et une liste `[b1; b2; ...; bn]: 'b list`, et renvoie `f(...f(f a b0) b1) ... bn)`. En d'autres termes, `fold_left` doit avoir le même comportement que la fonction `List.fold_left`. Par exemple :

   `(fold_left (fun a b -> a +. float_of_int b) 1. [2; 3; 4])` vaut `10.`

   `(fold_left (fun a b -> not a || b) false [false; true])` vaut `true`

4. Écrire une fonction `(for_all: ('a -> bool) -> 'a list -> bool)` qui prend en argument un prédicat `(p: 'a -> bool)` ainsi qu'une liste, et renvoie `true` si et seulement si `(p a)` vaut `true` pour tous les éléments de la liste. En d'autres termes, `(for_all p [a1; a2; ...; an])` vaut `(p a1) && (p a2) && ... && (p an)`.
5. Écrire une fonction `(exists: ('a -> bool) -> 'a list -> bool)` qui prend en argument un prédicat `(p: 'a -> bool)` ainsi qu'une liste, et renvoie `true` si et seulement s'il existe un élément de la liste `a` tel que `(p a)` vaut `true`.
6. Si vous n'avez pas utilisé la question 3 dans les questions 4 et 5, refaire ces questions en utilisant la fonction `fold_left`.
7. À l'aide de la fonction `fold_left`, écrire une fonction `(length: 'a list -> int)` qui renvoie la taille de la liste passée en argument. En d'autres termes, `length` doit avoir le même comportement que la fonction `List.length`.
8. À l'aide de la fonction `fold_left`, écrire une fonction `(iter_bis: ('a -> unit) -> 'a list -> unit)` comme dans la question 2.