Lycée Saint-Louis TP
MPSI

Opérations élémentaires sur les listes en Caml

L'objectif de ce TP est d'écrire des fonctions Ocaml simples qui manipulent les listes telles qu'elles sont implémentées dans le module List. D'après le programme, les fonctions `map`, `mem`, `for_all`, `exists` et `filter` doivent être connues et utilisables après rappel. On s'interdira bien entendu de les utiliser dans ce TP, à l'exception des constructeurs et de la longueur.

Pour chaque question, on commencera par déterminer le type de la fonction ou de l'objet à définir. On prouvera la terminaison, la correction et la complexité de la solution proposée.

1. Écrire une fonction `nombre_occurrences` prenant en argument une liste l et un élément x et renvoyant le nombre d'occurrences de x dans l.
2. Écrire une fonction `appliquer` prenant en argument une fonction f et une liste l et renvoyant la liste des images par f des éléments de l.
   Cette fonction existe déjà sous le nom de `List.map`.
3. Écrire une fonction `maximum` renvoyant le maximum des éléments d'une liste d'entiers non vide, et qui lève une exception si on l'appelle avec la liste vide.
4. Écrire une fonction `dernier` renvoyant le dernier élément d'une liste non vide, et qui lève une exception si on l'appelle avec la liste vide.
5. Écrire une fonction `contient` prenant en argument un élément a et une liste l et testant si a est dans l.
   Cette fonction existe déjà sous le nom `List.mem`.
6. Écrire une fonction `indice` prenant en argument un élément a et une liste l et renvoyant un indice, s'il existe, où apparaît a dans l, et qui lève une exception sinon.
7. Écrire une fonction `listeindice` prenant en argument un élément a et une liste l et renvoyant la liste des indices où apparaît a dans l.
8. Écrire une fonction `nieme` prenant en argument une liste l et un entier n et une liste l et renvoyant le n-ième élément de l, si la liste a au moins n éléments, et lève une exception sinon.
   Cette fonction existe déjà sous le nom `List.nth`.
9. Écrire une fonction `pour_tout` prenant en argument un prédicat p (c'est-à-dire une fonction de type de retour booléen) et une liste l et qui teste si tous les éléments de l vérifient p.
   Cette fonction existe déjà sous le nom `List.for_all`.
10. Écrire une fonction `existe` prenant en argument un prédicat p et une liste l et qui teste s'il existe un élément de l vérifiant p.
    Cette fonction existe déjà sous le nom `List.exists`.
11. Écrire une fonction `trouver` prenant en argument un prédicat p et une liste l et renvoyant le premier élément de l vérifiant p, s'il existe, et qui lève une exception sinon.
    Cette fonction existe déjà sous le nom `List.find` et lève l'exception `Not_found` si aucun élément de la liste ne vérifie le prédicat.
12. Écrire une fonction `filtre` prenant en argument un prédicat p et une liste l et renvoyant la liste des éléments de l vérifiant p.
    Cette fonction existe déjà sous le nom `List.filter`.
13. Écrire une fonction `partition` prenant en argument un prédicat p et une liste l et renvoyant le couple formé par la liste des éléments de l vérifiant p et la liste des éléments de l ne vérifiant pas p.
    Cette fonction existe déjà sous le nom `List.partition`.
14. Écrire une fonction `separer` prenant en argument une liste de couples et renvoyant le couple de listes correspondant.
    Cette fonction existe déjà sous le nom `List.split`.
15. Écrire une fonction `iterees` prenant en argument un élément a, une fonction f et un entier n, et renvoyant la liste à $n+1$ éléments $[a, f(a), f(f(a)), \ldots]$.
16. Écrire une fonction `entiersC` prenant en argument un entier n et renvoyant la liste des entiers de 0 à n, dans l'ordre croissant.