TP n°10 – Listes en OCaml

Testez vos fonctions sur des exemples de listes, y compris []

On rappelle qu'une liste, de type `'a list` :

- contient des éléments tous de même type `'a` ;
- Soit est la liste vide `[]`,
- soit possède une tête (premier élément) et une queue (liste des autres éléments), auxquels on accède par filtrage :

```OCaml
match liste with
| [] ->
| tete :: queue ->
```

- Les écritures `[1; 2; 3]` et `1 :: 2 :: 3 :: []` sont équivalentes.

Des squelettes de code sont disponibles pour la plupart des fonctions (pas les deux premières). Référez-y vous en cas de doute.

1 Fonctions de base sur les listes

Exercice 1

Écrire une fonction `longueur : 'a list -> int` qui renvoie la longueur d'une liste sans utiliser `List.length`. Quel est le nombre d'appels récursifs en fonction de la taille de la liste ?

Exercice 2

Écrire une fonction `tete : 'a list -> 'a` puis une fonction `queue : 'a list -> 'a list` qui renvoient respectivement la tête et la queue d'une liste. Si la liste est vide, on pourra utiliser l'exception `failwith "la liste est vide"` pour faire une erreur.

Exercice 3

Écrire une fonction `nieme : int -> 'a list -> 'a` qui, à partir d'un entier $n \geqslant 0$ et d'une liste, renvoie le n-ième élément de la liste. Si la liste ne possède pas de n-ième élément (parce qu'elle est trop courte), on pourra utiliser une exception.

Exercice 4

La concaténation de deux listes est l'opération qui "colle" deux listes ensemble, par exemple pour `l1 = [1; 2]` et `l2 = [3; 4; 5]` cela donne `[1; 2; 3; 4; 5]`.

Attention, l'ordre est important, les éléments de `l1` doivent apparaitre avant les éléments de `l2` dans le résultat.

On veut écrire une fonction `append : 'a list -> 'a list -> 'a list` qui renvoie la concaténation de deux listes sans utiliser `@`.

1. Si l_1 est vide, que vaut la concaténation de l_1 et l_2 ?
2. Imaginons maintenant que l_1 est de la forme $t :: q$.
   Notons l_3 la concaténation de q et l_2, comment obtenir la concaténation de l_1 et l_2 à partir de t et l_3 ?
3. Dans une logique récursive, comment obtenir l_3 ?
4. Écrire la fonction.

Quel est le nombre d'appels récursifs en fonction des tailles des deux listes ? Est-ce que la taille de l_2 a une importance ?

2 Quelques autres fonctions sur les listes

Exercice 5

Écrire une fonction `maximum : 'a list -> 'a` qui renvoie l'élément maximal d'une liste (d'éléments comparables). Si la liste est vide on lèvera une exception.

Exercice 6 Quel est le type de la fonction suivante, et que fait-elle ? Vérifier avec CAML.

OCaml

```
let rec mystere f liste =
match liste with
| [] -> []
| tete :: queue -> (f tete) :: mystere f queue
;;
```

Exercice 7 Écrire une fonction **`decouple : ('a * 'b) list -> 'a list * 'b list`** qui à partir d'une liste de couples renvoie un couple de listes.

Par exemple pour **`[(0,1);(3,4);(7,10)]`** on renvoie **`([0;3;7],[1;4;10])`**.

Exercice 8 Écrire une fonction **`for_all : ('a -> bool) -> 'a list -> bool`** prenant en argument un prédicat et une liste, et renvoyant **`true`** si le prédicat est vérifié pour tous les éléments de la liste, **`false`** sinon.

Un prédicat est une fonction qui a un élément de type **`'a`** associe un booléen

Par exemple :
- **`for_all (function x -> x > 2) [1; 2; 3]`** renvoie **`false`**
- **`for_all (function x -> x > 2) [4; 7; 9]`** renvoie **`true`**.

3 Quelques algorithmes de tri

Trier est un problème fondamental en terme d'algorithmique. On veut à partir d'une liste, obtenir une liste avec les mêmes éléments, mais rangés dans un certain ordre.

Dans la suite on triera toutes les données dans l'ordre croissant.

Exercice 9

Définir six listes différentes qui vous serviront de tests pour les fonctions de tri.

On s'assurera d'avoir la liste vide, une liste à un élément, une liste déjà triée, une liste d'éléments égaux, une liste avec des doublons et une liste sans doublons pour observer le comportement du tri sur chacune.

Exercice 10

On se propose de programmer le tri insertion (*insertion sort*) sur les listes. Le principe de ce tri est celui que vous utilisez pour trier un jeu de cartes en main : si vous avez une main triée et qu'on vous donne une nouvelle carte, vous pouvez trouver où l'insérer pour que votre main reste triée.

1. Écrire une fonction `insere : 'a -> 'a list -> 'a list` prenant en argument un objet et une liste triée par ordre croissant et renvoyant la liste obtenue en insérant l'objet dans la liste en conservant son caractère trié. Par exemple, **`insere 5 [2; 4; 7; 8; 9]`** renvoie **`[2; 4; 5; 7; 8; 9]`**.
2. Écrire une fonction `tri_insertion : 'a list -> 'a list` triant la liste : lorsque c'est encore possible, il suffit d'insérer la tête dans la queue qui aura été triée récursivement.

Exercice 11

L'idée du tri par sélection (ou tri par extraction) est d'extraire le minimum d'une liste, de trier récursivement la liste sans son minimum puis de placer ce minimum en tête de la liste. Implémenter ce tri en CAML.

4 Les ensembles

Dans cette partie on va utiliser une liste pour représenter un ensemble.

Les ensembles et les listes ont deux différences principales :
- Les ensembles sont sans doublons, donc vos listes devront l'être aussi.
- Les ensembles ne sont pas linéaires, et donc pas ordonnés. Vos fonctions n'ont donc pas besoin de préserver l'ordre des listes, mais si elles le font ce n'est pas grave.

1. Définir l'ensemble vide dans une variable **`ensemble_vide`**.
2. Écrire une fonction qui teste si un élément x est dans un ensemble. La signature de votre fonction sera **`cherche : 'a list -> 'a -> bool`**
3. Écrire une fonction qui fait l'intersection des deux ensembles.
4. Écrire une fonction qui fait l'union de deux ensembles. (avez vous pensé aux doublons ?)
5. Écrire une fonction qui fait la différence entre deux ensembles.
6. Écrire une fonction qui teste l'inclusion d'un ensemble dans un autre.
7. Écrire une fonction qui calcule le produit cartésien de deux ensembles.