Listes en OCaml

Florian Bourse

Listes

En OCaml, le type `'a list` est un type polymorphe représentant une liste ordonnée d'éléments de type `'a`. Il est défini par induction à l'aide de 2 constructeurs :

- `[]` représente une liste vide ;
- `t::q` représente une liste dont l'élément de tête est `t`, et la queue de la liste est `q`. `t` est de type `'a` et `q` de type `'a list`.

On peut aussi définir une liste de plusieurs éléments comme ceci : `[x1;x2;...xn]` qui est un sucre syntaxique pour l'expression `x1::x2::...::xn::[]`.

Les listes étant non-mutables en OCaml, les fonctions opérant sur les listes créent une nouvelle liste qu'elles renvoient.

Filtrage par motif — Pattern matching

Le filtrage par motif est une opération qui permet d'effectuer une disjonction de cas, afin de traiter un type énuméré/somme/union comme le type `'a list`. La syntaxe du filtrage par motif est constituée des mots-clés `match ... with` pour désigner la valeur qui sera filtrée, puis d'une définition de fonction `p0 -> v0 | p1 -> v1 | ...`, syntaxe que l'on retrouve pour filtrer les exceptions, et pour définir les fonctions avec le mot clé `fun`.

Exemple d'utilisation du pattern matching

```
let f x = match x with
  (* Le premier | n'est pas nécessaire *)
  | 5 -> 5
  (* On peut associer une même valeur à plusieurs motifs *)
  | 0 | 1 -> 1
  | 2 -> 0
  (* On peut nommer une variable dans un filtrage *)
  (* On peut ajouter une garde (condition) à un motif *)
  | n when n < 0 -> failwith "Le nombre donné doit être positif"
  | n when n mod 2 = 0 -> n / 2
  (* wildcard, _ accepte toutes les valeurs *)
  | _ -> 3
(* Seul l'expression du premier motif vérifié est évaluée *)
;;
```

Lors de la définition d'une fonction, on peut aussi utiliser le mot-clé `function` qui remplace `match ... with` avec la syntaxe suivante :

```
let f x = match x with
  | ...
(* equivaut a *)
let f = function
  | ...
```

Exemple de fonction sur les listes

Calcul de la taille d'une liste

```
let rec length l = match l with
  | [] -> 0
  | t::q -> 1 + length q;;
```

Cette fonction de complexité linéaire.

Quelques fonctions utiles sur les listes

Question 1. Écrire une fonction `somme : int list -> int` qui calcule la somme des éléments d'une liste.

Question 2. Écrire une fonction `carres : int list -> int list` qui renvoie la liste des carrés des éléments d'une liste.

Question 3. Écrire une fonction `range : int -> int -> int list` telle que `range i j` renvoie `[i;i+1;...;j]`, ou `[]` si $j < i$.

opérateur infixe `--`

```
# let (--) = range;;
val ( -- ) : int -> int -> int list = <fun>
# 4--10;;
- : int list = [4; 5; 6; 7; 8; 9; 10]
```

Question 4. Écrire une fonction `mem : 'a -> 'a list -> bool` qui teste l'appartenance d'un élément à une liste.

Question 5. Écrire une fonction `nth : 'a list -> int -> 'a` qui renvoie le n-ième élément d'une liste ou lève l'exception `Liste_vide` si il n'existe pas.

définition de l'exception pour les listes vides

```
exception Liste_vide;;
```

Question 6. Écrire une fonction `rev_append : 'a list -> 'a list -> 'a list` telle que `rev_append l1 l2` inverse l'ordre des éléments de `l1` et les ajoute en tête de `l2`.
Bonus : récursivité terminale

Question 7. Écrire une fonction `rev : 'a list -> 'a list` qui inverse l'ordre des éléments d'une liste, à l'aide de la fonction précédente.

Question 8. Écrire une fonction `append : 'a list -> 'a list -> 'a list` qui concatène deux listes à l'aide des deux fonctions précédentes.
Remarque : l'opérateur @ existe en OCaml pour concaténer des listes, mais n'est pas récursif terminale, donc moins efficace que votre fonction.

Question 9. Écrire une fonction `position : 'a list -> 'a -> int` telle que `position x l` renvoie la première position à laquelle apparaît la valeur `x` dans la liste `l`. Lèce l'exception `Not_found` si `x` n'appartient pas à `l`.

Fonctions d'ordre supérieure

Question 10. Écrire une fonction `exists : ('a -> bool) -> 'a list -> bool` telle que `exists p l` vérifie si il existe un élément $x \in \ell$ tel que $p(x)$ est vraie.
En déduire une autre écriture de la fonction `mem`.

Question 11. Écrire une fonction `for_all : ('a -> bool) -> 'a list -> bool` telle que `for_all p l` vérifie si pour tout élément $x \in \ell$, $p(x)$ est vraie.
En déduire une autre écriture de la fonction `mem`.

Question 12. Écrire une fonction `iter : ('a -> unit) -> 'a list -> unit` telle que `iter f [a1;...;an]` applique la fonction f à a_1, puis à a_2, etc. jusque a_n, de manière équivalente à `begin f a1; f a2; ...; f an; () end`.
En déduire une fonction `print_int_list` permettant l'affichage de listes d'entiers.

Question 13. Écrire une fonction `map : ('a -> 'b) -> 'a list -> 'b list` telle que `map f [a1;...;an]` renvoie la liste `[f a1; f a2; ... f an]`.
En déduire une autre écriture de la fonction `carres`

Question 14. Que fait la fonction suivante :

```
let myst l =
  let lp = map string_of_int l in
  iter print_endline lp;;
```

Question 15. Écrire une fonction `fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a` telle que `fold_left f init [b1; ...; bn]` vaut `f (... (f (f init b1) b2) ...) bn`.
En déduire une autre écriture de `exists`, `for_all`, `iter`, `map`, et `somme`.

Un peu de statistiques

On souhaite calculer la valeur médiane d'une liste d'entiers.

En utilisant un tableau d'effectifs

On va représenter un tableau d'effectifs de la manière suivante : si notre série statistique est représentée par une liste de type `'a list`, notre tableau d'effectifs sera représenté par le type `('a * int) list`, dans laquelle nous retrouveront toutes les valeurs possibles rangées dans l'ordre croissant, couplées avec leur effectif. En voici un exemple :

série statistique et tableau d'effectifs associé

```
let serie = [4;5;6;4;5;4;3;1;5;6;4];;
let tab = [(1,1);(3,1);(4,4);(5,3);(6,2)];;
```

Question 16. Écrire une fonction

`ajoute_valeur : 'a -> ('a * int) list -> ('a * int) list`

qui ajoute une valeur dans un tableau d'effectifs.

Question 17. Écrire une fonction `eff_tab : 'a list -> ('a * int) list * int` qui prend en entrée une série statistique et qui renvoie le tableau d'effectifs associé ainsi que l'effectif total de la série.

Question 18. Écrire une fonction `mediane : 'a list -> 'a` qui prend en entrée une série statistique et qui en renvoie la médiane.

Trier puis prendre le milieu

Question 19. Écrire une fonction `trie : 'a list -> 'a list` qui permet de trier une liste. En déduire une fonction `mediane : 'a list -> 'a`.