DS 1

Informatique MP2I

Julien Reichert

Toutes les questions de programmation sont à résoudre en OCaml.

Question de cours 1 : Comment accède-t-on au dernier élément d'un tableau en OCaml ?

Question de cours 2 : À quoi servent les mots-clés `rec` et `and` en OCaml ?

Exercice 1 : On cherche à calculer a^b en travaillant exclusivement avec des entiers. Écrire une fonction `puissance a b` de signature `int -> int -> int` réalisant ce calcul, sans la moindre conversion. On se limitera au cas où `b` est positif ou nul, sans besoin de vérifier ceci.

Exercice 2 : En fait, le calcul de a^b peut être accéléré en remarquant que $a^b = (a^2)^{\frac{b}{2}}$ et qu'on reste avec des entiers si b est pair. Pour des valeurs impaires de b, on peut se ramener à $b-1$ en procédant à un moment à une multiplication par a. Écrire une formule mathématique présentant la récurrence attendue avec une distinction de cas, et en déduire une fonction récursive procédant à cette exponentiation rapide, où le nombre de multiplications est limité au nombre de chiffres binaires composant b. La même restriction sur la positivité de b s'applique.

Exercice 3 : Écrire une fonction de signature `recolle : 'a array -> 'a array -> 'a array` prenant en argument deux tableaux et retournant un nouveau tableau formé de tous les éléments du premier puis de tous les éléments du deuxième.

Par exemple, `recolle [|2; 5; 4|] [|3; 1|]` devra retourner `[|2; 5; 4; 3; 1|]`.

Exercice 4 : On rappelle que la fonction `Array.init` permet d'initialiser un tableau à l'aide d'une taille et d'une fonction qui s'appliquera à chacun des indices. Ainsi, `Array.init 4 carre` donnera `[|0; 1; 4; 9|]` si on a écrit `let carre x = x * x;;`. Écrire une fonction ayant la même spécification... bien entendu sans se servir de la fonction elle-même. Attention à ne pas utiliser un nom avec un point dedans pour une fonction qu'on définit soi-même !

Exercice 5 : La fonction `List.init`, de même principe mais produisant une liste, a été ajoutée récemment au module `List` d'OCaml. La fonction `List.make` n'existe cependant toujours pas. Écrire deux fonctions de signatures `listinit : int -> (int -> 'a) -> 'a list` et `listmake : int -> 'a -> 'a list` ayant les spécifications attendues. Une récursion, directe ou indirecte, sera obligatoire dans les deux cas, et l'utilisation de références et boucles interdite.

Exercice 6 (difficile) : L'extraction de sous-tableaux en OCaml est limitée : les éléments sont forcément consécutifs, et déborder mène à une erreur. Écrire une fonction de signature `slice : 'a array -> int -> int -> int -> 'a array` telle que `slice tab d f p` retourne un sous-tableau du tableau `tab` formé des éléments aux positions de `d` inclus à `f` exclu par pas de `p`, avec les conditions suivantes : on impose que `d` soit compris entre `0` et la taille de `tab` moins un, que `f` soit compris entre `-1` (pour pouvoir aller jusqu'au début dans le cas d'un pas négatif) et la taille de `tab` (pour pouvoir aller jusqu'à la fin dans le cas d'un pas positif), et que `p` soit non nul, tous les arguments sont obligatoires et le résultat est vide si le pas ne va pas dans le bon sens par rapport aux seuils. Ainsi, on imite une version restreinte de `tab[d:f:p]` de Python.

Exercice 7 : Faire les preuves de terminaison et correction des fonctions des exercices 1 et 2. Calculer aussi les complexités en fonction des valeurs des arguments.

Problème 1 : On dispose d'un tableau de flottants tous différents, et on souhaite en déduire l'information du classement de chacun des éléments **dans l'ordre décroissant**. Pour ceci, le protocole sera le suivant : [étape 1] créer un tableau reprenant les mêmes flottants dans le même ordre mais au sein de triplets où les deux autres valeurs sont pour l'une l'indice dans le tableau et pour l'autre une valeur initialement mise à -1, [étape 2] trier le tableau par un appel pertinent à une fonction de tri, [étape 3] remplacer le troisième élément de chaque triplet par l'indice auquel les valeurs sont désormais dans le tableau trié, [étape 4] trier à nouveau le tableau selon l'indice ayant existé précédemment et restant stocké dans le triplet, [étape 5] créer le tableau final dont les éléments sont des couples formés par le flottant et l'indice dans la version triée en rejetant l'indice dans la version de départ qui est une information redondante. Écrire une fonction (appelant éventuellement des fonctions écrites en-dehors pour certaines étapes) de signature `classement : float array -> (float * int) array` réalisant le travail demandé. On notera qu'a priori la signature peut aussi remplacer les deux occurrences du type `float` par `'a`. Il faudra bien se poser la question de la mutabilité ou non des objets manipulés (en parler explicitement) et de l'accès à des données d'un couple et d'un triplet.

Par exemple, pour le tableau de départ `[|54.2; 45.6; 34.6; 78.5; 45.9; 99.3|]`, l'étape 1 crée le tableau

```
[|(54.2, 0, -1); (45.6, 1, -1); (34.6, 2, -1); (78.5, 3, -1); (45.9, 4, -1); (99.3, 5, -1)|],
```

l'étape 2 mute ce tableau, pour qu'il devienne

```
[|(99.3, 5, -1); (78.5, 3, -1); (54.2, 0, -1); (45.9, 4, -1); (45.6, 1, -1); (34.6, 2, -1)|],
```

l'étape 3 le mute aussi pour qu'il devienne

```
[|(99.3, 5, 0); (78.5, 3, 1); (54.2, 0, 2); (45.9, 4, 3); (45.6, 1, 4); (34.6, 2, 5)|],
```

l'étape 4 le mute une dernière fois pour qu'il devienne

```
[|(54.2, 0, 2); (45.6, 1, 4); (34.6, 2, 5); (78.5, 3, 1); (45.9, 4, 3); (99.3, 5, 0)|]
```

et l'étape 5 produit la réponse attendue, à savoir

```
[|(54.2, 2); (45.6, 4); (34.6, 5); (78.5, 1); (45.9, 3); (99.3, 0)|].
```

On offre la fonction `Array.sort` dont la spécification est la suivante :

```
Array.sort : ('a -> 'a -> int) -> 'a array -> unit
```

Trie un tableau dans l'ordre croissant selon une fonction de comparaison. La fonction de comparaison doit retourner `0` si ses arguments sont à considérer comme égaux, un entier positif si le premier argument est à considérer comme supérieur et un entier négatif si le premier est à considérer comme inférieur.

La fonction de comparaison doit représenter une relation antisymétrique et transitive, et une fois la fonction de tri selon `comparaison` terminée sur le tableau `tab`, pour tous les indices `i` et `j` valides, `comparaison tab.(i) tab.(j)` est positif ou nul si et seulement si `i` est supérieur ou égal à `j`.

Problème 2 : On rappelle certains exercices du DM 1 ci-après. Les fonctions suivant les énoncés correspondent à des réponses dans le désordre. Attribuer à chaque réponse son exercice en justifiant autant que possible. **Les exercices en question ne sont pas à traiter !**

[13] Écrire une fonction `coupe` prenant en argument une liste et retournant un couple de listes contenant dans le même ordre les éléments de la liste fournie répartis alternativement en commençant par la première liste du couple.

[14] Écrire une fonction `colle` prenant en argument deux listes dont on supposera qu'elles sont de même taille et retournant la liste obtenue en prenant alternativement un élément de chaque liste.

[15] Écrire une fonction `compte` prenant en argument une liste et une valeur du type des éléments de la liste et retournant le nombre d'occurrences de la valeur dans la liste.

[16] Écrire une fonction `sommesi` prenant en argument une liste d'entiers et un prédicat sur les entiers (c'est-à-dire une fonction prenant en entier et retournant un booléen) et retournant la somme des éléments de la liste vérifiant le prédicat (donc celui-ci renvoie vrai) et seulement eux.

[17] Écrire une fonction `majopred` prenant en argument une liste et un prédicat sur les éléments de la liste et retournant un booléen déterminant si au moins la moitié des éléments de la liste vérifient le prédicat.

[18] Écrire une fonction `lexico` prenant en argument deux listes et retournant un booléen indiquant si la première est inférieure à la deuxième dans un ordre lexicographique (le premier élément strictement inférieur à l'élément à la même position entre les listes détermine le résultat final, et si une liste est un préfixe de l'autre elle y est inférieure).

[19] Écrire une fonction `aplatir` prenant en argument une liste de listes et retournant la liste contenant tous les éléments de ces listes à la suite.

```
let somme li = List.fold_left (+) 0 li;; (* Ceci n'est pas une solution d'exercices
mais sert par la suite ! *)

let f0 li p = somme (List.filter p li);;

let f1 li x = let compteur c n = c + (if x = n then 1 else 0) in List.fold_left compteur 0 li;;

let f2 li p = let compteur n a = if p a then n + 1 else n - 1 in List.fold_left compteur 0 li >= 0;;

let f3 li p = let compteur n a = n + (if p a then a else 0) in List.fold_left compteur 0 li;;

let f4 li x = let compare_x n = if x = n then 1 else 0 in somme (List.map compare_x li);;

let f5 li = let arob l1 l2 = l1 @ l2 in List.fold_left arob [] li;;
```

On rappelle les spécifications des fonctions `map`, `filter` et `fold_left` du module `List` :

`List.map : ('a -> 'b) -> 'a list -> 'b list` renvoie la liste des images par la fonction en premier argument des éléments de la liste en deuxième argument, dans le même ordre.

`List.filter : ('a -> bool) -> 'a list -> 'a list` renvoie la liste formée par les éléments de la liste en deuxième argument qui satisfont un prédicat en premier argument.

`List.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a` renvoie l'application imbriquée de la fonction en premier argument aux éléments de la liste en troisième argument avec la valeur par défaut en deuxième argument (dont le type peut différer de celui des éléments de la liste, mais en accord avec la signature de la fonction à appliquer de manière imbriquée).
En clair, `List.fold_left f b [a1; ...; an]` correspond à `f (...(f (f b a1) a2) ...) an`.