

TD/TP 2b : programmation fonctionnelle, OCaml

TRÈS IMPORTANT : vérification des programmes

Testez systématiquement tous les exemples OCaml qui ont été donnés en TD. Testez, avec encore plus de soin, les petits programmes OCaml que vous avez vous-même écrit. Vous devez considérer comme faux un programme que vous n'avez pas testé.

Partie 1 : récursivité

Exercice 1.1 (modélisation de Σ)

Le calcul de la somme d'une série de la forme :

$$\sum_{i=m}^n f(i) = f(m) + f(m+1) + \dots + f(n)$$

se traduit en OCaml par la fonction récursive suivante :

```
# let rec somme f m n = if m > n then 0
                        else (f m) + (somme f (m+1) n) ;;
(* f : argument de type fonction *)
(* m : borne inférieure de l'intervalle *)
(* n : borne supérieure de l'intervalle *)
```

Quel est le type de `somme` ? Expliquer votre réponse.

Donner un **exemple** d'appel de la fonction `somme` pour chacun des calculs suivants :

1. La somme des entiers de 1 à n .
2. La somme des entiers compris dans l'intervalle $[m, n]$.
3. La somme des cubes (x^3) des entiers compris dans l'intervalle $[m, n]$.
4. La somme des factorielles compris dans l'intervalle $[1, n]$.

Exercice 1.2 (récursivité sur les chaînes de caractères)

Les fonctions outils pour la réalisation de l'exercice :

```
# let car ch = String.get ch 0 ;;
```

```
(* retourne le premier caractère de ch *)
val car : string -> char = <fun>

# let cdr ch = String.sub ch 1 ((String.length ch) - 1);;
(* retourne ch sans son premier élément *)
val cdr : string -> string = <fun>;

# cdr "azerty";;
- : string = "zerty";
(* nous utilisons cdr pour parcourir la chaîne *)
```

Nombre d'occurrences d'une chaîne

Nous souhaitons compter le nombre de fois qu'apparaît la chaîne « 00 » dans une chaîne donnée. L'énoncé ci-dessous est une description de ce calcul.

$$occur(y) = \begin{cases} 0, & \text{si } y = "" \\ 1 + occur(cdr(cdr(y))), & \text{si } x = "00" \\ occur(cdr(y)), & \text{sinon} \end{cases}$$

où x est une sous chaîne extraite de y à partir de sa première position. Une réalisation possible de cet énoncé par un programme OCaml est :

```
# let rec occur y =
  match y with
  | "" -> 0
  | _ -> let ch1 = cdr y in
        let ch2 = cdr ch1 in
        if String.sub y 0 2 = "00"
        then 1 + occur ch2
        else occur ch1 ;;
```

Ce programme ne fait pas ce qu'il est censé faire. Corriger-le. Pour cela vous pouvez vous aider de la commande « trace ».

1. Tracer l'exécution de « `occur "120004009";;` » comme suit :

```
# #trace occur;;
# occur "120004009";;
```

2. Combien de fois la fonction `occur` est-elle appelée ?
3. Décrire l'étape de calcul du dernier appel de la fonction, c'est-à-dire remplacer y par « 9 » dans `occur` et en déduire l'erreur.
4. Corriger la fonction `occur`.

Écrire une fonction « `miroir` » de type « `string -> string` » qui calcule l'image miroir d'une chaîne de caractères.

Exemples :

- `miroir "ONU"` renvoie `"UNO"`,

- miroir "eluparcettecrapule" renvoie "eluparcettecrapule".

Cette fonction doit utiliser les fonctions « car » et « cdr ». Vous pouvez utiliser la fonction « Char.escaped c » pour convertir un caractère c en une chaîne d'un caractère afin d'appliquer l'opération de concaténation.

Exemple :

```
# let a = "azerty" ;;
a : string = "azerty";;
# let b = a^Char.escaped 'e';;
b : string = "azertye";;
```

Exercice 1.3 (les nombres premiers)

Question 1 :

Écrire une fonction booléenne récursive `div_between` telle que `div_between a b n` détermine s'il existe un entier c entre a et b inclus tel que c divise n.

Question 2 :

À l'aide de la fonction précédente, écrire une fonction `is_prime` qui détermine si un nombre donné en argument est premier.

Question 3 :

Commencez par définir une fonction `no_div_from` telle que `no_div_from a n` renvoie true si et seulement si n n'a pas de diviseurs différents de n plus grands que a. Nous ferons attention à ne pas tester tous les entiers.

Question 4 :

À l'aide de la fonction précédente, écrire une nouvelle fonction `is_prime` qui détermine si un nombre donné en argument est premier.

Exercice 1.4 (récursivité double)

Ecrire une fonction qui calcule les coefficients du binôme, $\binom{n}{p}$ pour n et p entiers naturels, par la méthode récursive simple telle que décrite ci-dessous.

$$\text{binome}(p, n) = \begin{cases} 1, & \text{si } p = 0 \text{ ou } p = n \\ \text{binome}(p, n-1) + \text{binome}(p-1, n-1), & \text{sinon} \end{cases}$$