Exercice 1 : Modélisation de Σ

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

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

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 *)
```

ou encore

```
# let rec somme f m n = if m < n then 0
                 else (f n) + (somme f m (n-1));;
```

Quel est le type de `somme` ?

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 2 : Récursivité sur les chaînes de caractères

On rappelle les fonctions vues en TD4 :

```
# let car ch = String.get ch 0 ;; (* retourne le premier car. de ch *)
val car : string -> char = <fun>
# let cdr ch = String.sub ch 1 ((String.length ch) - 1);;
(* retourne ch sans son premier element *)
val cdr : string -> string = <fun>;
# cdr "azerty";;
- : string = "zerty";
(* on utilise cdr pour parcourir la chaîne *)
```

1 : Nombre d'occurences d'une chaîne

On veut compter le nombre de fois qu'apparaît la chaîne "00" dans une chaîne donnée. L'énoncé ci-dessous est une description de ce calcul.

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

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. On vous demande de le corriger. Pour cela on peut s'aider de la commande trace.

1. tracer l'execution 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`

2 : L'inverse d'une chaîne

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`. La fonction `car` retourne un caractère. 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" ;;
# let b = a^Char.escaped 'e';;
b : string = "azertye";;
```

Exercice 3 : Récursivité double

Écrire 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.

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