Feuille d'exercices n°2

Cette feuille d'exercice porte sur la définition de fonctions par récurrence sur un entier. Même lorsque ce n'est pas explicitement demandé, vous êtes invités à proposer aussi des versions récursives terminales des fonctions.

Exercice I : Factorielle

La factorielle d'un nombre entier est définie comme :

$$\begin{cases} 0! &= 1 \\ n! &= n \times (n-1)! \quad \text{si } n > 0 \end{cases}$$

Q1 – Définir une fonction récursive de signature

```
fact (n:int) : int
```

telle que `(fact n)` donne la valeur de `n`!.

Q2 – Donner une version récursive terminale de la fonction factorielle.

Exercice II : Sommes

Q1 – Définir la fonction de signature

```
sum_n (n:int) : int
```

qui calcule la somme des n premiers nombres entiers strictement positifs.

Par exemple: `(sum_n 5)` $= 5+4+3+2+1 = 15$ et `(sum_n 0)` $= 0$.

Q2 – En utilisant une fonction locale, redéfinir la fonction `sum_n` de manière à ce qu'elle déclenche l'exception `Invalid_argument "sum_n"` si `n` est négatif.

Q3 – Définir la fonction de signature `sum_imp (n:int) : int` qui calcule la somme des n premiers nombres impairs positifs.

Par exemple: `(sum_imp 4)` $= 7+5+3+1 = 16$. C'est-à-dire $(4\times2-1)+(3\times2-1)+(2\times2-1)+(1\times2-1)+0$

On aura également que `(sum_imp n)` $= 0$ si `n` ≤ 0.

Q4 – **Un peu plus difficile:** définir la fonction de signature `sum_f (f:int -> int) (n:int) : int` qui calcule la somme `(f n) + (f (n-1)) + ... + (f 0)`.

Par exemple, si `succ` est la fonction qui associe $n+1$ à l'entier n, alors :

`(sum_f succ 4) = (succ 4)+(succ 3)+(succ 2)+(succ 1)+(succ 0) = 5+4+3+2+1 = 15`

Q5 – Utilisez la fonction `sum_f` pour définir la fonction `sum_f_imp` qui calcule la même fonction `sum_imp`: pour tout `n:int`, `(sum_f_imp n)` = `(sum_imp n)`.

Exercice III : Termes d'une suite

Soit u_n définie par: $\begin{cases} u_0 &= 42 \\ u_{n+1} &= 3u_n + 4 \end{cases}$

Q1 – Donnez la définition de la fonction `u : int -> int` telle que `(u n)` donne $u_{\mathtt{n}}$. Quelle hypothèse faut-il faire sur `n` ?

Q2 – En utilisant la fonction `u`, donnez la définition de la fonction `sum_u` qui calcule $\Sigma_{i=0}^{n} u_n$, c'est-à-dire, la somme des $n+1$ premiers termes de la suite u_n.

Q3 – Redéfinir `sum_u` en utilisant la fonction `sum_f` de l'exercice précédent.

Q4 – **Plus difficile:** définir `sum_u` sans utiliser la fonction `u`.

1. Sans utiliser la fonction `u`, définir une fonction de signature `loop (n:int) (t:int) : int` qui calcule la somme `t + v(t) + v`^{2}`(t) + ... + v`^{n}`(t)` où `v(x)` est égal à $3\mathtt{x}+4$.
   Indication: l'argument `t` est modifié à chaque itération : `((3 * t) + 4)`.
2. En déduire une nouvelle définition de `sum_u`.
3. Faire de `loop` une définition locale pour définir `sum_u`.

Exercice IV : Récurrence sur un intervalle

Q1 – Donnez une définition de la fonction de signature `sum_inter (a:int) (b:int) : int` qui donne la somme des entiers compris dans l'intervalle $[\mathtt{a}, \mathtt{b}]$. Par convention, la somme vaut 0 si l'intervalle est vide. L'intervalle $[\mathtt{a}, \mathtt{b}]$ est vide lorsque $\mathtt{a} > \mathtt{b}$.

Faut-il poser des hypothèses sur les arguments ? Élaborez un jeu de tests pour cette fonction.

Q2 – Définir la fonction de signature `sum1_inter (k:int) (a:int) (b:int) : int` qui donne la somme `(k+a) + (k+(a+1)) + .. + (k+b)`.

Q3 – **Plus difficile:** donnez la définition de la fonction de signature
`sum2_inter (a:int) (b:int) : int` qui donne la somme de tous les couples d'entiers de l'intervalle $[\mathtt{a}, \mathtt{b}]$.

Par exemple: `(sum2_inter 2 4)` = (2+2)+(2+3)+(2+4)+(3+2)+(3+3)+(3+4)+(4+2)+(4+3)+(4+4)

Indication: utiliser une fonction récursive locale qui utilise `sum1_inter`.

Exercice V : Itérateur – À faire à la maison

On définit l'itérateur suivant (un peu différent de celui vu en cours):

```
let rec iter2 (n:int) (f:int -> 'a -> 'a) (b:'a) : 'a =
  if (n > 0) then (f n (iter2 (n-1) f b))
  else b
```

Schématiquement `(iter2 n f b)` = `(f n (f (n-1) ... (f 1 b) ...))`;
et, si `n` ≤ 0 alors `(iter2 n f b)` = `b`.

Par exemple, si `f` est la fonction d'addition `fun i r -> i+r`, `(iter2 n f 0)` donne la somme `n + (n-1) + .. +1 + 0`. On peut ainsi (re)définir `sum_n` avec `iter2`.

Q1 – Utilisez `iter2` pour définir la fonction `fact_i` telle que:
pour tout `n:int`, `(fact_i n)` = `(fact n)`.

Q2 – Utilisez `iter2` pour définir la fonction `sum_imp_i` telle que:
pour tout `n:int`, `(sum_imp_i n)` = `(sum_imp n)`.

Q3 – Utilisez `iter2` pour définir la fonction `sum_f_i` telle que:
pour tout `n:int`, `(sum_f_i n)` = `sum_f n`.

Q4 – Utilisez `iter2` pour définir la fonction `sum_inter_i` telle que:
pour tout `n:int`, `(sum_inter_i n)` = `sum_inter n`.