TP 2 d'option informatique - MPSI - 05/02/2021

Exercice 1. Type d'une fonction

1. Sur une feuille, donner le type de chacune des fonctions suivantes ainsi que le résultat des appels à ces fonctions. Vérifier votre réponse avec OCaml.

```
let rec add n m =
  if n = 0 then
    m
  else
    add (n-1) (m+1);;
```

```
add 0 10;;
add 5 10;;
add 5 (-1);;
add (-4) (-1);;
```

```
let rec f x = f x;;
f 0;;
```

2. De même avec les fonctions anonymes suivantes :

```
(fun a -> a + a*a) 2;;
fun f x y -> f x y;;
fun f g x -> g (f x);;
fun f g x -> (f x) + (g x);;
fun x y z -> (x y) z;;
```

Exercice 2.

On souhaite écrire une fonction `(double: int -> int)` qui prend en entrée un entier $n \in \mathbb{N}$ et renvoie $2n$.

1. Sans utiliser Caml, expliquez pourquoi la fonction suivante ne convient pas. Si vous ne trouvez pas, demandez à Caml d'afficher les valeurs de `n` pour chaque appel récursif.

```
let rec double_fail n = match n with
  | n when n < 0 -> failwith "n < 0"
  | n -> 2 + double_fail (n-1)
  | 0 -> 0;;
```

2. En Caml, écrire une version corrigée de `double`.
3. En utilisant la fonction `double` mais sans utiliser `+` ou `*`, écrire une fonction `(fois_huit: int -> int)` qui prend en entrée un entier $n \in \mathbb{N}$ et renvoie $8n$. Testez votre fonction.

Exercice 3. Suites numériques

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par :

$$\begin{cases} u_0 = 0 \\ u_{n+1} = 2u_n^3 + 1 \qquad \text{pour tout } n \in \mathbb{N} \end{cases}$$

1. Écrire une fonction récursive de type `(int -> int)` qui prend en argument un entier $n \in \mathbb{Z}$ et renvoie u_n ou une erreur si n est strictement négatif. Vérifiez que votre fonction répond instantanément pour $n = 20$.

Soit $(v_n)_{n\in\mathbb{N}}$ et $(w_n)_{n\in\mathbb{N}}$ les suites définies par :

$$\begin{cases} v_0 = 0 \\ v_{n+1} = v_n + w_n \end{cases} \quad \text{pour tout } n \in \mathbb{N} \qquad \begin{cases} w_0 = 1 \\ w_{n+1} = \left\lfloor \dfrac{v_n}{w_n} \right\rfloor + 1 \end{cases} \quad \text{pour tout } n \in \mathbb{N}$$

2. Écrire une fonction récursive de type `(int -> int * int)` qui prend en argument un entier $n \in \mathbb{Z}$ et renvoie le couple (v_n, w_n).
3. En utilisant la question précédente, écrire une fonction de type `(int -> int)` qui prend en argument un entier $n \in \mathbb{Z}$ et renvoie v_n. De même pour la suite $(w_n)_{n\in\mathbb{N}}$.

La syntaxe ci-dessous permet de définir deux fonctions `f` et `g` par récursivité croisée. En d'autres termes, le corps de la fonction `f` peut faire des appels récursifs à `g` et vice-versa.

```
let rec f n =
  <corps de la fonction f>
and g m =
  <corps de la fonction g>
```

4. Refaire la question 3 à l'aide d'une récursivité croisée (et sans utiliser les questions précédentes).

On définit la suite de Fibonacci $(f_n)_{n\in\mathbb{N}}$ par :

$$\begin{cases} f_0 = 0 \\ f_1 = 1 \\ f_{n+2} = f_{n+1} + f_n \end{cases} \quad \text{pour tout } n \in \mathbb{N}$$

5. Écrire une fonction récursive qui prend en argument un entier naturel $n \in \mathbb{N}$ et renvoie f_n.
6. Essayer de calculer f_{50}. Expliquez l'origine du problème que vous rencontrez.
7. En utilisant la même idée que dans la question précédente, dire pourquoi l'implémentation de la question 4 n'est pas satisfaisante alors que celle de la question 3 l'est.

On souhaite donc résoudre le problème rencontré dans la question 6.

8. Écrire une fonction récursive `(fibo_aux: int -> int * int)` qui prend en argument un entier n et renvoie le couple (f_n, f_{n+1}).
9. En utilisant la question 8, refaire la question 5 et vérifier que le problème rencontré à la question 6 n'apparaît plus.

Exercice 4.

Soit $f : \mathbb{Z} \times \mathbb{N} \to \mathbb{Z}$ et $g : \mathbb{Z} \times \mathbb{N} \to \mathbb{Z}$ définies par :

$$\begin{cases} f(n,0) = n \\ f(n,m) = f(n,m-1) + 1 \end{cases} \qquad \begin{cases} g(n,0) = n \\ g(n,m) = g(n+1,m-1) \end{cases}$$

1. Écrire les fonctions f et g en OCaml.
2. Trouver une formule close pour f et pour g (c'est à dire une formule non récursive).

Exercice 5. Manipulation de références

1. Écrire une fonction de type `('a ref -> 'a ref -> unit)` qui échange le contenu de deux références.
2. Déterminer le type et l'utilité de la fonction suivante :

```
let f x y =
  x := !x + !y;
  y := !x - !y;
  x := !x - !y;;
```

Exercice 6. Exemples de fonctions

Pour chacune des questions suivantes, vous devez écrire deux fonctions : une qui utilise un paradigme itératif et l'autre qui utilise un paradigme récursif.

1. Écrire une fonction de type `(int -> int -> int)` qui calcule le PGCD de deux nombres à l'aide de l'algorithme d'Euclide.
2. Écrire une fonction qui prend en entrée deux entiers p et n, et renvoie $S_n = \sum_{k=1}^{n} p^k$.

   Indication : exprimer S_n en fonction de S_{n-1}, d'une addition et d'une multiplication.
3. Le symétrique d'un entier positif $n \geqslant 0$ est obtenu en écrivant les chiffres de n à l'envers. Par exemple, le symétrique de 123 est 321. Sans utiliser de conversion de type, écrire une fonction qui renvoie le symétrique de l'entier donné en paramètre.
4. Écrire une fonction qui prend en entrée deux entiers $p \geqslant 0$ et n, et renvoie $\sum_{k=1}^{n} k^p$.

Exercice 7. Exponentiation rapide

Dans cet exercice, vous n'avez pas le droit d'utiliser la puissance `**` de Ocaml.

1. Écrire une fonction récursive (`pow: float -> int -> float`) qui prend en entrée un réel x ainsi qu'un entier naturel n, et renvoie x^n. Votre fonction affichera un message d'erreur lorsque $n < 0$ ou lorsque $x = n = 0$.
2. Écrire une fonction `pow2` qui prend en entrée un réel x et un entier **relatif** n et renvoie x^n. Votre fonction affichera un message d'erreur lorsque le calcul est impossible.

Pour $x \in \mathbb{R}^*$ et $n \in \mathbb{N}^*$, le principe de l'exponentiation rapide est d'utiliser les relations suivantes :

$$x^0 = 1, \qquad x^n = \left(x^{n/2}\right)^2 \text{ si } n \text{ est pair}, \qquad x^n = x \times \left(x^{(n-1)/2}\right)^2 \text{ si } n \text{ est impair.}$$

3. Écrire une fonction `pow_rapide` qui prend en entrée un réel x ainsi qu'un entier **relatif** n et renvoie x^n calculé à l'aide d'une exponentiation rapide. Vous n'avez pas le droit d'utiliser `**`. Vérifiez que votre programme termine instantanément pour $x = 1.000\ 000\ 001$ et $n = 1\ 000\ 000\ 000$.
4. Écrire une exponentiation rapide à l'aide d'une fonction itérative.

Exercice 8. Décomposition en facteurs premiers

1. Écrire une fonction itérative (`print_decomp: int -> unit`) qui affiche la décomposition d'un entier $n \in \mathbb{N}$ en facteurs premiers. Par exemple :

   `print_decomp (-1)` affiche une erreur, `print_decomp 0` affiche `0`,
   `print_decomp 1` affiche `1`, `print_decomp 2` affiche `2`,
   `print_decomp 4` affiche `2^2`, `print_decomp 2520` affiche `2^3 3^2 5 7`,
   `print_decomp 790614` affiche `2 3^3 11^4`.

2. De même en récursif.

Exercice 9. Parité des coefficients binomiaux

Théorème de Lucas : *Soit $n, k \in \mathbb{N}$ avec $n \geqslant k$ deux entiers que l'on écrit en base 2 :*

$$n = (n_1 n_2 \dots n_p)_2, \qquad k = (k_1 k_2 \dots k_p)_2.$$

Le coefficient binomial $\binom{n}{k}$ est impair si et seulement si :

$$\forall i \in \{1, \dots, p\} : \Big[k_i = 1 \Rightarrow n_i = 1\Big].$$

1. En admettant le théorème de Lucas, écrire une fonction récursive `binom_est_pair` qui prend en entrée n et k, et qui renvoie `true` lorsque $\binom{n}{k}$ est pair et `false` sinon. Testez votre fonction pour toutes les valeurs de n et k avec $n, k \in [\![0, 20]\!]$.
2. De même avec une fonction itérative.

Exercice 10. Nombres de Catalan

1. Écrire une fonction (`somme: (int -> int) -> int -> int`) qui prend entrée une fonction f ainsi qu'un entier n et renvoie l'entier $\displaystyle\sum_{i=0}^{n} f(i)$.
2. À l'aide de la fonction `somme`, écrire une fonction (`catalan: int -> int`) qui prend en entrée un entier n et renvoie le nombre de Catalan c_n où :

$$\begin{cases} c_0 = 1 \\ c_1 = 1 \\ c_n = \displaystyle\sum_{p+q=n-1} c_p c_q \qquad \text{Si } n \geqslant 2 \end{cases}$$

3. Vérifiez à l'aide de Caml que pour tout $n \in [\![0, 10]\!]$:

$$c_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\, n!} = \prod_{i=2}^{n} \frac{n+i}{i}.$$

Exercices à rendre au plus tard le 10/02/2021

Les exercices doivent m'être envoyés à l'adresse `rendu.opt.mpsi1@gmx.fr` ou bien `rendu.opt.mpsi2@gmx.fr` en fonction de votre classe. Attention, ce n'est pas la même adresse mail que pour l'informatique commune.

Exercice 11. Ensemble défini récursivement

Soit $A \subseteq \mathbb{N}$ le plus petit ensemble tel que :

- A contient 0 et 1
- Si $a \in A$ alors $2a \in A$ et $5a + 6 \in A$

Écrire une fonction `(contient: int -> bool)` qui renvoie `true` lorsque l'entier donné en paramètre appartient à A et `false` sinon. Par exemple :

`contient 0` vaut `true`,	`contient 1` vaut `true`,	`contient (-1)` vaut `false`,
`contient 126` vaut `true`,	`contient 216` vaut `false`,	`contient 472` vaut `true`,
`contient 1926` vaut `true`,	`contient 2345` vaut `false`.	

Exercice 12. Fonction itérée

Écrire une fonction `iteree` qui prend en entrée une fonction `(f: 'a -> 'a)` et un entier `n` positif ou nul et renvoie la fonction itérée $f^{(n)} : x \mapsto f \circ \ldots \circ f(x)$. Testez votre fonction.

Exercice 13. Suite de Goodstein (exercice facultatif)

La suite de Goodstein est une suite d'entiers naturels dont le comportement est contre-intuitif : malgré le fait que la suite semble croissante et que les premiers termes sont rapidement gigantesques, la suite est nulle à partir d'un certain rang.

Pour définir la suite de Goodstein, on a besoin de la notion de *notation héréditaire*. Soit $b \geqslant 2$ et $u \geqslant 0$ deux entiers naturels. La notation héréditaire en base b se définit récursivement de la manière suivante :

- 0 est la notation héréditaire de $u = 0$.
- Si $u \neq 0$ alors pour écrire la notation héréditaire de u en base b :
  - On écrit u en base b.
  - On écrit les exposants de cette décomposition en notation héréditaire en base b.

Par exemple, la notation héréditaire de $u = 415$ en base $b = 3$ est $u = 3^{3^{3^0}+2\times 3^0} + 2 \times 3^{3^{3^0}+3^0} + 3^{2\times 3^0} + 3^0$ car :

$$\begin{aligned} u &= 415 \\ &= 3^5 + 2 \times 3^4 + 3^2 + 3^0 \\ &= 3^{3^1+2\times 3^0} + 2 \times 3^{3^1+3^0} + 3^{2\times 3^0} + 3^0 \\ &= 3^{3^{3^0}+2\times 3^0} + 2 \times 3^{3^{3^0}+3^0} + 3^{2\times 3^0} + 3^0. \end{aligned}$$

Soit $u \in \mathbb{N}$ un entier naturel. La suite de Goodstein de graine u est la suite $(u_n)_{n\in\mathbb{N}^*}$ où $u_1 = u$ et pour tout $n \geqslant 2$:

- Si $u_{n-1} = 0$ alors $u_n = 0$.
- Sinon, u_n est obtenu à partir de u_{n-1} de la manière suivante :
  - On écrit la notation héréditaire de u_{n-1} en base n.
  - On remplace tous les n de la notation héréditaire de u_{n-1} par des $n+1$.
  - On enlève 1 au nombre obtenu.

Par exemple, lorsque $u = 4$, on a :

- $u_1 = u = 4 = 2^{2^{2^0}}$ donc $u_2 = 3^{3^{3^0}} - 1 = 26$
- $u_2 = 26 = 2 \times 3^{2 \times 3^0} + 2 \times 3^{3^0} + 2 \times 3^0$ donc $u_3 = 2 \times 4^{2 \times 4^0} + 2 \times 4^{4^0} + 2 \times 4^0 - 1 = 41$
- $u_3 = 41 = 2 \times 4^{2 \times 4^0} + 2 \times 4^{4^0} + 1 \times 4^0$ donc $u_4 = 2 \times 5^{2 \times 5^0} + 2 \times 5^{5^0} + 1 \times 5^0 - 1 = 60$
- ...

Écrire une fonction qui prend en entrée la graine u ainsi qu'un entier n, et qui renvoie u_n. Vérifiez que pour $u = 3$:

$$u_1 = 3, \qquad u_2 = 3, \qquad u_3 = 3, \qquad u_4 = 2, \qquad u_5 = 1, \qquad u_6 = 0, \qquad u_7 = 0.$$

Et que pour $u = 10$:

$$u_1 = 10, \qquad u_2 = 83, \qquad u_3 = 1025, \qquad u_4 = 15625, \qquad u_5 = 279935.$$

Exercices à rendre au plus tard le 17/02/2021

Exercice 14. Type d'une fonction

1. Sur une feuille, donner le type de la fonction suivante ainsi que le résultat des appels à cette fonction. Justifiez.

```
let rec f a = match a with
  | 0 -> 0
  | _ when a < 0 -> 1 + f (a+1)
  | _ -> 1 + f (a-1);;
```

```
f 0;;
f (-5);;
f 16;;
```

2. De même avec les fonctions anonymes suivantes :

```
fun x y z -> x (y z);;
fun x y z -> x y z;;
fun x y z -> x (y z x);;
fun x y z -> (x y) (z x);;
fun x y -> (x y) + (y x);;
```

Exercice 15. Approximation de la racine carrée avec l'algorithme de Babylone

Soit $x \in \mathbb{R}_+$ et $(u_n)_{n\in\mathbb{N}}$ la suite définie par :

$$\begin{cases} u_0 = 1 \\ u_{n+1} = \dfrac{1}{2}\left(u_n + \dfrac{x}{u_n}\right) \text{ pour tout } n \in \mathbb{N} : \end{cases}$$

La suite $(u_n)_{n\in\mathbb{N}}$ converge alors vers $\sqrt{x}$.

1. Écrire une fonction itérative (`racine: float -> float -> float`) qui prend en entrée x et ϵ et renvoie le premier terme u_n tel que $\left|u_n - \dfrac{x}{u_n}\right| < \epsilon$.
2. De même en récursif.

Exercice 16. Partition et composition d'un entier (exercice facultatif)

Une partition d'un entier $n \in \mathbb{N}$ consiste à écrire n comme une somme d'entiers strictements positifs. Par exemple $n = 5$ admet 7 partitions :

$5 = 1+1+1+1+1$, $5 = 1+1+1+2$, $5 = 1+2+2$, $5 = 1+1+3$,
$5 = 2+3$, $5 = 1+4$, $5 = 5$.

Dans une partition, l'ordre des termes de la somme n'est pas important, par exemple $5 = 2+3$ et $5 = 3+2$ représentent la même partition.

1. Écrire une fonction (`nb_partitions: int -> int -> int`) qui prend en entrée deux entiers n et m et renvoie le nombre de partions de n avec m termes. Vérifiez les valeurs pour $n = 5$ et que :

   `nb_partitions 5 0` vaut `0`, `nb_partitions 5 6` vaut `0`,
   `nb_partitions 8 4` vaut `5`, `nb_partitions 10 5` vaut `7`,
   `nb_partitions 20 2` vaut `10`, `nb_partitions 20 10` vaut `42`.

2. Écrire une fonction (`nb_total_partitions: int -> int`) qui prend en entrée un entier n et renvoie le nombre de partions de n. Par exemple :

   `nb_total_partitions 5` vaut `7`, `nb_total_partitions 10` vaut `42`,
   `nb_total_partitions 15` vaut `176`, `nb_total_partitions 20` vaut `627`.

Une composition d'un entier $n \in \mathbb{N}$ consiste à écrire n comme une somme d'entiers, l'ordre des termes étant important. Par exemple, $n = 5$ admet 16 compositions :

$5 = 1+1+1+1+1$, $5 = 1+1+1+2$, $5 = 1+1+2+1$, $5 = 1+2+1+1$,
$5 = 2+1+1+1$, $5 = 1+2+2$, $5 = 2+1+2$, $5 = 2+2+1$,
$5 = 1+1+3$, $5 = 1+3+1$, $5 = 3+1+1$, $5 = 2+3$,
$5 = 3+2$, $5 = 1+4$, $5 = 4+1$, $5 = 5$.

3. Écrire une fonction `(nb_compositions: int -> int -> int)` qui prend en entrée deux entiers n et m et renvoie le nombre de compositions de n avec m termes. Vérifiez les valeurs pour $n = 5$ et que :

   `nb_compositions 5 0` vaut `0`, `nb_compositions 5 6` vaut `0`,
   `nb_compositions 8 4` vaut `35`, `nb_compositions 10 5` vaut `126`,
   `nb_compositions 20 2` vaut `19`, `nb_compositions 20 10` vaut `92378`.

4. Écrire une fonction `(nb_total_compositions: int -> int)` qui prend en entrée un entier n et renvoie le nombre de compositions de n. Vérifiez avec Caml que pour tout $n \in [\![1, 10]\!]$, le nombre de compositions de n vaut 2^{n-1}.