TD 2 Programmation fonctionnelle

1 Échauffement

Écrire une fonction `pgcd` donnant le plus grand diviseur commun de ses deux arguments (entiers positifs), en se basant sur l'algorithme d'Euclide :

$$pgcd(a,b) = \begin{cases} a & \text{si } a = b \\ pgcd(a-b,b) & \text{si } a > b \\ pgcd(a,b-a) & \text{si } b > a \end{cases}$$

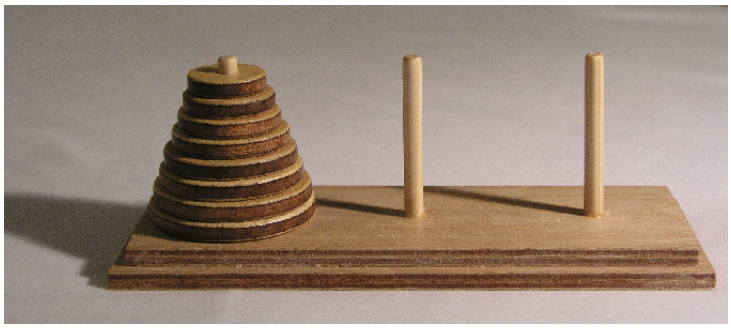
.

Solution.

```
let rec pg n m =
  if n=m then n
  else if n<m then pg n (m-n)
  else pg (n-m) m
```

□

2 La tour de Hanoi



Vous devez déplacer la tour d'un pilier vers un autre en déplaçant un disque à la fois sans jamais placer un disque au-dessus d'un disque plus petit.

Solution.

```
let bouge_disque origine destination =
(*deplace un seul disque de l'origine vers la destination*)
print_string "Deplacer un disque du pilier ";
print_int origine;
print_string " vers le pilier ";
print_int destination;
```

```
print_endline ".";;
# val bouge_disque : int -> int -> unit = <fun>

let rec hanoi initial terminal auxiliaire disques =
(*Tours Hanoi de initial vers terminal en utilisant auxiliaire *)
if disques = 1
then bouge_disque initial terminal
else begin
hanoi initial auxiliaire terminal (disques-1);
bouge_disque initial terminal;
hanoi auxiliaire terminal initial (disques-1)
end;;
# val hanoi : int -> int -> int -> int -> unit = <fun>
```

□

3 Jouer avec les fonctions

1. Définissez la fonction `min` qui calcule le minimum entre deux nombres. Son type doit être `int -> int-> int`.

   Solution.

   ```
   let min x y = if x < y then x else y
   ```

   □

2. Déduisez-en la fonction `plafonne_a n` qui renvoie une fonction f de type `int -> int` telle que `f m` vaut m plafonné à n. Le type de `plafonne_a` est le même que celui de `min`.

   Solution.

   ```
   let plafonne_a n = fun m -> min n m

   (* peut aussi s'écrire*)
   let plafonne_a n m = min n m

   (* ou encore, avec une évaluation partielle *)
   let plafonne_a n = min n

   (* ou encore, plus simplement *)
   let plafonne_a  = min
   ```

   □

3. Définissez une fonction `permute_args f` qui prend en argument une fonction f ayant pour arguments a puis b et qui renvoie la fonction ayant pour arguments b puis a. Indication : le type de `permute_args` doit être `('a -> 'b -> 'c) -> 'b -> 'a -> 'c`.

   Solution.

   ```
   let permute_args f a b = f b a

   (* ou  *)

   let permute_args f = fun a b -> f b a
   ```

□

4. En réutilisant les fonctions pair et impair vu en classe, écrire la fonction qui donne la longueur d'un vol de la suite de Syracuse partant de la valeur n :

$$syracuse(n) = \begin{cases} 0 & \text{si } n = 1 \\ 1 + syracuse(n/2) & \text{si n pair} \\ 1 + syracuse(1 + 3 * n) & \text{si n impair} \end{cases}$$

Solution.

```
let rec
  pair n = if n = 0 then true else impair (n-1)
and
  impair n = if n= 0 then false else pair (n-1);;

let rec syracuse n =
  if n = 1 then 0
  else if pair n then 1 + syracuse (n/2)
  else 1 + syracuse (1 + (3*n));;
```

□