TD 1

1 Typage et évaluation

Exercice 1 Donner le type des expressions suivantes :

1. `fun x -> x + 1`
   Solution : `int -> int`
2. `fun x y -> x *. y`
   Solution : `float -> float -> float`
3. `fun x (y, z) -> if x then y else not z`
   Solution : `bool -> (bool * bool) -> bool`
4. `fun x -> x`
   Solution : `'a -> 'a`
5. `fun x y z -> (x <= y, z)`
   Solution : `'a -> 'a -> 'b -> (bool * 'b)`

Exercice 2 Soit la fonction `mystere` suivante, donner le résultat de `mystere 3`.

```
let rec mystere x =
  if x = 0 then [x] else x :: mystere (x - 1)
```

Solution : on concatène tous les éléments de 0 à x ⇒ `mystere 3 -> [3; 2; 1; 0]`

2 Boucles sans boucles

Exercice 3 Écrire à l'aide de fonctions récursives en OCaml les programmes suivants (écrits en langage C) :

(1)

```
int x = 0;
while (x <= 4) {
  printf("%d", x);
  x = x + 2;
};
```

(2)

```
int x = 0;
int y = 10;
while (x <= 4) {
  printf("%d", y - x);
  x = x + 2;
  y--;
};
```

(3)

```
int x = 0;
int y = 5;
int z = 100;
while (x <= y) {
  printf("%d\n", z);
  x++;
  y = y - x;
};
```

Solution de (1) :

```
let rec f x =
  if x <= 4 then
    (Printf.printf "%d " x ;
     f (x+2))
;;
f 0;;
```

Solution de (2) :

```
let rec f x y =
  if x <= 4 then
    (Printf.printf "%d " (y-x) ;
     f (x+2) (y-1))
;;
f 0 10;;
```

Solution de (3) :

```
let rec f x y z =
  if x <= y then
    (Printf.printf "%d\n" z ;
     f (x+1) (y-x) z)
;;
f 0 5 100;;
```

Exercice 4 Écrire le programme suivant à l'aide d'une fonction récursive en OCaml :

```
int acc = 0;
for (int i = 0; i<=3; i++){
 acc = acc + 2;
};
printf("%d", acc);
```

Solutions :

Version renvoyant le résultat

```
let rec f x acc =
  if x <= 3 then f (x+1) (acc+2)
  else acc
;;

Printf.printf "%d" (f 0 0);;
```

Version affichant directement le résultat

```
let rec f x acc =
  if x  <= 3 then f (x+1) (acc+2)
  else Printf.printf "%d" acc
;;

f 0 0;;
```

3 Récursion terminale et CPS

Exercice 5 Soit la fonction `somme_carre`, de type `int -> int`, telle que `somme_carre n` renvoie la somme des carrés de 1 à `n`, c'est-à-dire $1 + 2^2 + 3^2 + \cdots + n^2$. Écrire une version récursive terminale de cette fonction.

Solution :

```
let somme_carre n =
  let rec sc_acc acc n =
    if n <= 0 then failwith "negative n"
    else if n = 1 then acc+1
    else sc_acc (acc + n*n) (n-1)
  in
  sc_acc 0 n
;;
```

Exercice 6 Écrire une version récursive terminale et une version par CPS de la fonction `somme` suivante :

```
let rec somme l =
  match l with
  | []     -> 0
  | x :: s -> x + somme s
```

Solutions :

Récursive terminale :

```
let somme l =
  let rec s_acc acc l =
    match l with
    | []    -> acc
    | x::tl -> s_acc (acc+x) tl
  in
  s_acc 0 l
;;
```

CPS :

```
let somme l =
  let rec s_cps l cont =
    match l with
    | []    -> cont 0
    | x::tl -> s_cps tl (fun r -> cont (r+x))
  in
  s_cps l (fun x -> x)
;;
```

Exercice 7 Écrire une version par CPS de la fonction de Fibonacci suivante :

```
let rec fib n =
  if n <= 1 then n
  else fib (n-1) + fib (n-2)
```

Solution :

```
let fib n =
  let rec fib_cps n cont =
    if n <= 1 then cont n
    else
      fib (n-1) (fun res1 -> fib (n-2) (fun res2 -> cont (res1+res2)))
  in
  fib_cps n (fun x -> x)
;;
```