Corrigé du TP 1 de programmation fonctionnelle en Objective Caml

Christian Rinderknecht

3 février 2014

1 Nombres

```
let p x = 2 * x*x + 3 * x - 2
let q x = 2.0 *. x*.x +. 3.0 *. x -. 2.0
```

2 Expressions conditionnelles

```
# if -3 < 0 then 3 else 3;;
- : int = 3
# let abs_int n = if n < 0 then -n else n;;
val abs_int : int -> int = <fun>
```

3 Evaluation d'une expression

Pas de questions.

4 Déclarations

4.1 Variables globales

```
# let an = "2003";;
val an : string = "2003"
# let x = int_of_string(an);;
val x : int = 2003
# let nouvel_an = string_of_int(x+1);;
val nouvel_an : string = "2004"
```

4.2 Variables locales

```
# let x = 3 in let b = x < 10 in if b then 0 else 10;;
- : int = 0
# let a = 3.0 and b = 4.0 in sqrt(a*.a+.b*.b);;
- : float = 5.
# let solve (a,b,c) =
    let delta = b *. b -. 4.0 *. a *. c
```

```
  in ((-. b -. sqrt (delta))/. (2.0 *. a),
      (-. b +. sqrt (delta))/. (2.0 *. a));;
val solve : float * float * float -> float * float = <fun>
```

5 Produit cartésien

Pas de questions.

6 Récursivité

Pas de questions.

7 Expressions fonctionnelles

```
# let compose f g x = f (g(x));;
val compose : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
# let add1 = function x -> x + 1;;
val add1 : int -> int = <fun>
# let mult5 = function x -> 5 * x;;
val mult5 : int -> int = <fun>
# compose add1 mult5 3;;
- : int = 16
# compose mult5 add1;;
- : int -> int = <fun>
# let deuxfois f x = f(f(x));;
val deuxfois : ('a -> 'a) -> 'a -> 'a = <fun>
# let deuxfois f x = compose f f x;;
val deuxfois : ('a -> 'a) -> 'a -> 'a = <fun>
```

8 Portée statique des variables

```
# let p = 10;;
p : int = 10
# let k x = x + p;;
k : int -> int = <fun>
# let p = p+1;;
p : int = 11
# p;;
- : int = 11
# k 0;;
- : int = 10
```

La variable `p` qui apparaît dans la définition de la fonction `k`, est liée *définitivement* à la valeur `10`. La troisième phrase redéfinit une nouvelle variable `p`, rendant inaccessible la première variable `p`, *sauf pour la fonction* `k`. En effet, une fonction est modélisée par une *fermeture*, c'est-à-dire une paire constituée du code de la fonction et de l'environnement statique au lieu de définition.