Université Paris Diderot — Programmation Fonctionnelle
Licence — Année 2010-2011

TP n°1 - Correction

Premiers pas en OCaml

Les exercices marqués d'une étoile [⋆] peuvent être laissés pour la fin.

Exercice 1 Pour lancer l'interpréteur OCaml sous `emacs` :
- ouvrir un fichier `tp1.ml`
- dans le menu `Tuareg`, choisissez l'entrée `Run Caml Toplevel` qui se trouve dans le sous-menu `Interactive Mode`.
- confirmer le lancement de `ocaml`

Chaque expression entrée dans la fenêtre de `tp1.ml` peut être évaluée en se plaçant sur un caractère quelconque de l'expression, puis par : ou bien `Evaluate phrase` dans le sous-menu `Interactive Mode`" du menu `Tuareg` d'emacs, ou bien `ctrl-x, ctrl-e`.

Expressions et types

Exercice 2 Prévoir le résultat fourni par l'interpréteur OCaml après chacune des commandes suivantes, entrées dans l'ordre.

```
#let x = 2;;
#let x = 3
 in  let y = x + 1
      in x + y;;
#let x = 3 and y = x + 1
 in  x + y;;
```

Correction :

```
# let x = 2;;
val x : int = 2
# let x = 3
  in let y = x + 1
     in x + y;;
- : int = 7
# let x = 3 and y = x + 1
  in x + y;;
- : int = 6
```

La deuxième commande utilise la définition de l'identificateur x fraichement introduit (celle de $x = 3$), alors que la troisième commande utilise la première définition de x où elle vaut 2.

Pourquoi les deux dernières commandes ne fournissent-elles pas le même résultat ? Expliquer à présent le comportement suivant :

```
#let x = 3;;
x : int = 3
#let f y = y + x;;
```

```
f : int -> int = <fun>
#f 2;;
- : int = 5
#let x = 0;;
x : int = 0
#f 2;;
- : int = 5
```

Correction : La définition de f utilise la première définition de x. Redéfinir x dans la suite n'a aucun effet sur la définition de f qui est déjà fixée.

Exercice 3 Donner le type des expressions suivantes. Vérifier le résultat sur machine.
- `let f x = x + 1 in f 6;;`
- `let f x = x + 1 in f;;`
- `let s = "a" in let f t = t^s in f;;`

Correction :

```
# let f x = x + 1 in f 6;;
- : int = 7
# let f x = x + 1 in f;;
- : int -> int = <fun>
# let s = "a" in let f t = t^s in f;;
- : string -> string = <fun>
```

Booléens et conditionnelle

Les expressions de type `bool` sont de la forme :
- constantes booléennes :
  - `true`
  - `false`
- comparaisons :
  - e_1 *cmp* e_2

    où l'opérateur est `=`, `<>`, `<`, `<=`, `>`, ou `>=`, avec e_1 et e_2 de même type.
    On ne peut pas comparer des fonctions, mais on peut comparer des entiers, des chars, des listes, des n-uplets, etc.
- connections logiques :
  - e_1 `&&` e_2, e_1 `||` e_2, `not` e,

    (et, ou, négation de) avec e_1, e_2, e de type `bool`.

En OCaml, tout type non fonctionnel est implicitement muni d'un ordre, et la valeur d'une comparaison entre éléments de même type ne dépend que de cet ordre. Pour les types numériques, il s'agit de l'ordre usuel sur les nombres, pour les `char` l'ordre alphabétique, pour les `string` l'ordre lexicographique, etc.

L'expression suivante a pour valeur la valeur de e_1 si l'expression booléenne c vaut `true`, et celle de e_2 si c vaut `false` :

`if` c `then` e_1 `else` e_2

Exercice 4 Ecrire une fonction `est_majeur` prenant en argument une année de naissance (un entier), et renvoyant `true` si la différence entre l'année courante et cette année de naissance est au moins égale à 18, `false` sinon.

Correction :

```
let est_majeur a = 2010 - a >= 18;;
```

Exercice 5 Par convention, une année est bissextile : si elle est divisible par 4 et non divisible par 100, ou, si elle divisible par 400. En vous aidant de l'opérateur `mod` sur les entiers, écrivez la fonction `est_bissextile` qui prend comme argument une année (un entier), renvoyant `true` si elle et bissextile et `false` sinon.

Correction :

```
let est_bissextile a = (a mod 4 = 0 && (a mod 100 <> 0)) || a mod 400 = 0;;
```

Exercice 6 [⋆] La fonction `max : 'a -> 'a -> 'a` est l'une des fonctions prédéfinies. Appliquée à x et y de même type, elle renvoie le maximum de x et y (suivant l'ordre implicite sur les valeurs de ce type, voir ci-dessus).

Ecrire votre propre fonction `max`, en ne vous servant que de `if then else` et des opérateurs de comparaisons. Votre fonction doit être même type que le `max` prédéfini, c'est-à-dire de type `'a -> 'a -> 'a`.

A partir de cette fonction, définir :

- une fonction `max_couple` de type `` `a * `a -> `a ``, prenant en argument un couple `(x,y)`, et renvoyant le maximum de ses composantes,
- une fonction `max_triplet` de type `` `a * `a * `a -> `a `` prenant en argument un triplet `(x,y,z)`, et renvoyant le maximum de ses composantes.

Correction :

```
let max x y = if x > y then x else y;;
let max_couple  (x,y) = max x y;;
let max_triplet (x,y,z) = max x (max y z);;
```

Récurrence

La construction

`let rec` *nom arg_1* `=` *expression* `;;`

permet de se servir du nom d'une fonction dans sa propre définition, c'est-à-dire de définir des fonctions récursives. Par exemple :

```
let rec fact n = if n <= 0 then 1 else n*(fact (n - 1));;
```

Noter que `and` permet de définir simultanément plusieurs fonctions mutuellement récursives :

```
let rec f x = if x > 0 then (g (x - 1)) else 1
    and g x = if x > 0 then (f (x - 1)) else 0
;;
```

Exercice 7 Rappelons que la somme des entiers de 0 à n peut être définie récursivement par $\Sigma(0) = 0$, et $\Sigma(n) = n + \Sigma(n-1)$ si $n > 0$. Ecrire cette fonction en OCaml. Que donne cette fonction appliquée à un nombre négatif ?

Correction :

```
let rec somme n =
  if n <= 0 then 0 else n + (somme (n-1))
;;
(* n <= 0 au lieu de n = 0, pour eviter *)
(* un bouclage si n est negatif         *)
```

Exercice 8 Ecrire en OCaml la fonction récursive définie par $f(0) = 1$, $f(1) = 1$ et $f(n) = f(n-1) + f(n-2)$.

Correction :

```
let rec fibo n =
  if n <= 1 then 1
  else (fibo (n-1)) + (fibo (n-2))
;;
(* meme remarque *)
```

Exercice 9 [$\star$] Les fonctions en OCaml sont des valeurs comme les autres. En particulier, une fonction peut prendre en argument une ou plusieurs fonctions, et renvoyer une fonction :

```
let composer f g = (fun x -> f (g x));;
```

Ecrire une fonction prenant en argument une fonction `f` et un entier `n`, et renvoyant $\sum_{\mathtt{i}=0}^{\mathtt{n}} \mathtt{f}(\mathtt{i})$. Quel sera son type ? Utiliser cette fonction pour définir une nouvelle fonction calculant la somme des $n+1$ premiers carrés.

Correction :

```
let rec somme_f f n =
  if n < 0 then 0
  else (f n) + (somme_f f (n-1))
;;
let somme_carre = somme_f (fun x -> x * x)
;;
(* ou, ce qui revient au meme : *)
let somme_carre n = somme_f (fun x -> x * x) n
```

Exercice 10 [$\star$] Ecrire une fonction `binome` p n renvoyant C_n^p, donné par :

$$C_n^p = C_{n-1}^p + C_{n-1}^{p-1} \text{ si } 1 \leq p \leq n$$

$$C_n^0 = 1$$

$$C_n^p = 0 \text{ si } p > n$$

Correction :

```
(* en supposant n et p positifs : *)
let rec binome p n =
  if p = 0 then 1 else
  if p > n then 0 else
    binome p (n-1) + binome (p-1) (n-1)
;;
```