Programmation fonctionnelle : correction feuille d'exercices 1

María-Virginia Aponte

28 février 2011

Exercice 1

Expressions

Chacune des phrases Ocaml dans cet exercice est une *expression*. *Rappel* : Cela signifie en particulier que l'on peut appliquer des opérateurs ou des fonctions sur ces expressions ou les passer en paramètre.

On utilise ici les fonctions prédéfinies suivantes :
- `succ` : prend en argument un entier n en renvoie en résultat l'entier n+1 ;
- `Char.lowercase` : prend un caractère c et renvoie ce caractère convertit en majuscule ;
- `Char.code` : prend en argument un caractère et renvoie son code ASCII ;
- `Char.chr` : prend en argument un entier correspondant à un code ASCII et renvoie le caractère associé.

Question 1

1. Devinez les valeurs et types des expressions, puis comparez votre réponse avec celle donnée par Ocaml.
2. Classifiez ces expressions. S'agit-il d'une expression arithmétique, de comparaison, un appel de fonction, expression conditionnelle ?

Solution :

- expressions arithmétiques : lignes 1, 4, 19, 22, 25, 28, 37, 43, 58.
- expressions de comparaison (résultat booléen) : lignes 7, 10, 13, 49, 52, 55.
- expressions conditionnelles : 40
- appels de fonctions : 16, 22, 25, 31, 34, 37, 46, 61.

Question 2

Ces expressions sont mal typées. En vous aidant de la réponse donnée par Ocaml, expliquez pourquoi.

```
1  # 2 + 3.0;;
2  This expression has type float but is here used with type int
3
4  # succ 3.0;;
5  This expression has type float but is here used with type int
6
7  # 1 < 1.5;;
8  This expression has type float but is here used with type int
9
10 # if 4=4 then 1 else true;;
11 This expression has type bool but is here used with type int
```

```
12
13 # succ 3 4;;
14 Error: This function is applied to too many arguments;
15 maybe you forgot a ';'
```

Solution :

- ligne 1 : l'opérateur + ne peut être appliqué que sur des entiers.
- ligne 4 : l'argument de `succ` doit être de type `int`.
- ligne 7 : un opérateur de comparaison doit s'appliquer sur deux opérandes de même type, ce qui n'est pas le cas ici car `int` $\neq$ `float`.
- ligne 10 : les types des deux branches d'une conditionnelles doivent être identiques, or ici `int` $\neq$ `bool`.
- ligne 13 : la fonction `succ` prend un seul argument.

Exercice 2

Déclarations, expressions.

Classifiez les phrases suivantes en déclarations ou expressions. (*Rappel* : les déclarations locales sont des expressions). Expliquez les réponses données par Ocaml.

```
1  # let x = 2 in x+3;;
2  - : int = 5
3
4  # x+3;;
5  Unbound value x
6
7  # (let x=2 in x+3) + (let z=4 in z);;
8  - : int = 9
9
10 #  (let x=2 in x+3) + (let z=4 in z+x);;
11 Unbound value x
12
13 # let x= let  y=3 in y;;
14 val x : int = 3
15
16 # x;;
17 - : int = 3
18
19 # let cube x = x*x*x;;
20 val cube : int -> int = <fun>
21
22 # cube 2;;
23 - : int = 8
24
25 # let double_cube x = 2*(cube x);;
26 val double_cube : int -> int = <fun>
27
28 # double_cube 2;;
29 - : int = 16
```

Solution :

- expressions : lignes 1, 4, 7, 10, 13, 16, 19, 22, 28, 34.

- déclarations : lignes 25, 31.
- ligne 4 : x est déclarée localement, donc, non visible ici.
- ligne 10 : même explication, x est local à la première opérande de l'addition, donc non visible dans la deuxième.
- ligne 13 : le résultat de (`let y = 3 in y`) est 3. Donc, cette phrase équivaut à déclarer `let x = 3` globalement.
- ligne 16 : utilisation de la variable x déclarée ligne 13.
- ligne 19 : déclaration d'une fonction qui prend un entier et renvoie son cube. Son type est donc `int` $\rightarrow$ `int`
- ligne 22 : application de la fonction `cube` sur l'entier 2, le résultat est 2^3 qui est de type `int`.

Exercice 3

La fonction `String.length` calcule la longueur d'une chaîne de caractères. Quelles sont les réponses données par Ocaml ?

```
#  String.length "Bonjour" - 1;;
- : int = 6

# (String.length "Bonjour") - 1;;
- : int = 6

#String.length ("Bonjour" - 1);;
This expression has type string but is here used with type int

# let moyenne x y = (x+.y) /. 2.0;;
val moyenne : float -> float -> float = <fun>

# moyenne 4.0 6.0;;
- : float = 5.

# moyenne (4.0 6.0);;
This expression is not a function, it cannot be applied

#  let test x = if x>0 then 1 else "un";;
This expression has type string but is here used with type int
```

Exercice 4

Bien comprendre les expressions et les déclarations locales.

Question 1

Dans le code suivant il y a des phrases de syntaxe très proches mais pour lesquelles les messages Ocaml sont différents. Expliquez pourquoi.

```
1 # (if 3>7 then 1 else 10) + 5;;
2 - : int = 15
3
4 # let a = (if 3>7 then 1 else 10) + 5;;
5 val a : int = 15
6
7 # let b = 1 in let c = b;;
```

```
 8  Error: Syntax error
 9
10  # let b = 1 in let c = b in b=c;;
11  - : bool = true
12
13  # b;;
14  Error: Unbound value b
15
16  # let b = let c = 3 in c+5;;
17  val b : int = 8
18
19  # b;;
20  - : int = 8
```

Solution :

- la ligne 1 est une expression, la ligne 4 est une déclaration globable avec la valeur de cet expression.
- dans la ligne 7 il manque la partie "in".
- ligne 10 : C'est une déclaration locale imbriquée (donc une expression). On commence par déclarer `(let b=1)`, puis dans `(let c = b in b=c)`, c prend la valeur de b, et l'expression finale est la comparaison `(b=c)` qui renvoie true.
- ligne 13 : b est déclarée localement, donc non visible ici.
- ligne 16 : C'est une déclaration locale de b, avec la valeur de l'expression `(let c = 3 in c+5)`, égal à 8.

Question 2

Que fait la fonction `f` ? La fonction `g` est-elle locale ou globale ? Que fait-elle ? Testez la fonction `f` sur deux exemples différents.

```
# let f x =
  let g y = Char.code y
  in Char.chr (succ (g x));;
val f : char -> char = <fun>
```

Solution :

- la fonction g est locale à f et c'est donc une fonction auxiliaire. Elle prend un caractère et renvoie son code ASCII.
- la fonction f prend en argument un caractère. Elle utilise g pour calculer son code ASCII, et succ pour trouver le code suivant. Elle renvoie en résultat le caractère correspondant à ce nouveau code.
- Exemples d'utilisation :

```
# f 'a';;
- : char = 'b'
# f '3';;
- : char = '4'
# f 'C';;
- : char = 'D'
```

Exercice 5

Premières fonctions
Considérez les déclarations de fonctions suivantes :

```
# let minuscule c = ('a' <= c) && (c <= 'z');;
val minuscule : char -> bool = <fun>
```

```
# let majuscule c =  ('A' <= c) && (c <= 'Z');;
val majuscule : char -> bool = <fun>

# let lettre c = minuscule c || majuscule c;;
val lettre : char -> bool = <fun>
```

1. Que font ces fonctions ? Testez les dans la boucle interactive d'Ocaml en donnant à chacune successivement le caractères 'b', 'B', '*' et '4' en argument.

   Solution : `majuscule` teste si un caractère est une majuscule, `minuscule` fait autant pour les minuscules, et `lettre` teste si un caractère est une lettre.

2. Donnez une définition de la fonction `lettre` où les fonctions `majuscule` et `minuscule` sont déclarées localement (et simultanément).
   Solution :

   ```
   # let lettre c =
       let minuscule c = ('a' <= c) && (c <= 'z')
       and majuscule c =  ('A' <= c) && (c <= 'Z')
   in minuscule c || majuscule c;;
   val lettre : char -> bool = <fun>
   ```

3. Donnez une définition de la fonction `lettre` sans faire appel à d'autres fonctions.
   Solution :

   ```
   # let lettre c =
       ('a' <= c) && (c <= 'z') || ('A' <= c) && (c <= 'Z');;
   val lettre : char -> bool = <fun>

   # lettre 'a';;
   - : bool = true
   # lettre 'C';;
   - : bool = true
   # lettre '8';;
   - : bool = false
   ```

4. Donnez une définition d'une fonction `chiffre` qui teste si un caractère est un chiffre. Exemple : `chiffre '5'` répond true et `chiffre 'a'` répond false.

   Solution :

   ```
   # let chiffre c =
       ('0' <= c) && (c <= '9');;
    val chiffre : char -> bool = <fun>

   # chiffre 'a';;
   - : bool = false

   # chiffre '8';;
   - : bool = true
   ```

Exercice 6

Ecrire les premières fonctions
L'appel `(a mod b)` calcule le reste de la division entière de a par b. Par exemple :

```
# 7 mod 3;;
- : int = 1
# 6 mod 3;;
- : int = 0
# 2 mod 3;;
- : int = 2
```

1. Á l'aide de cette fonction, écrivez une fonction `divisible` qui prend deux entiers a et b, et teste si a est divisible par b. Testez votre fonction sur plusieurs exemples pour vous assurer qu'elle fonctionne correctement.
   Solution :

   ```
   # let divisible (a,b) = a >= b && a mod b = 0;;
   val divisible : int * int -> bool = <fun>

   # divisible (3,4);;
   - : bool = false
   #  divisible (3,3);;
   - : bool = true
   # divisible (9,3);;
   - : bool = true
   # divisible (10,3);;
   - : bool = false
   ```

2. Utilisez la fonction `divisible` pour écrire une fonction `pair` qui prend en argument un entier a et teste s'il est pair.
   Solution :

   ```
   # let pair a = divisible (a,2);;
   val pair : int -> bool = <fun>

   # pair 3;;
   - : bool = false
   # pair 2;;
   - : bool = true
   # pair 8;;
   - : bool = true
   ```

Exercice 7

Écrire les premières fonctions
L'appel `float_of_int n` convertit un entier n dans son équivalent de type `float`. Par exemple :

```
# float_of_int;;
- : int -> float = <fun>
# float_of_int 2;;
- : float = 2.
# 3.5 +. 2;;
This expression has type int but is here used with type float
# 3.5 +. (float_of_int 2);;
- : float = 5.5
```

1. Expliquez les messages donnés par Ocaml pour le code précèdent.
2. Utilisez la fonction `float_of_int` pour écrire une fonction `inverse` qui prend en argument un entier x et calcule son inverse $1/x$. Votre fonction doit calculer un `float`. Exemple : `inverse 2` doit répondre `- : float = 0.5`
   Solution :

```
# let inverse x = 1.0/. (float_of_int x);;
val inverse : int -> float = <fun>

# inverse 2;;
- : float = 0.5
```

3. Testez votre fonction avec l'argument 0. Expliquez le message affiché par Ocaml. **Solution** : La division par 0 est indéfinie :

```
#  inverse 0;;
- : float = infinity
```

Exercice 8

Écrire des fonctions
L'appel `int_of_char c` renvoie le code ASCII d'un caractère c (l'équivalent de Char.code). Par exemple :

```
# int_of_char 'a';;
- : int = 97
# int_of_char '0';;
- : int = 48
```

1. Expliquez les messages donnés par Ocaml pour le code précèdent. **Solution** : 97 et 48 sont respectivement les codes numériques des caractères 'a' et '0'.
2. Utilisez la fonction `int_of_char` pour écrire une fonction `car_vers_chiffre` qui prend en argument un caractère c qui est un chiffre et calcule l'entier qu'il représente. Si le caractère n'est pas un chiffre, la fonction renvoie son caractère unicode. Exemple : `car_vers_chiffre '2'` doit répondre `- : int = 2`, et `car_vers_chiffre 'a'` doit répondre `- : int = 97`.

```
# let car_vers_chiffre c =
    let chiffre c =     ('0' <= c) && (c <= '9')
  in  if chiffre c then int_of_char c - int_of_char '0'
      else int_of_char c;;
val car_vers_chiffre : char -> int = <fun>
```

3. Testez votre fonction sur plusieurs sortes de caractères.

```
# car_vers_chiffre '4';;
- : int = 4
#  car_vers_chiffre 'a';;
- : int = 97
```