Programmation fonctionnelle

TP 1

Giron David thor@epitech.net

Résumé: Dans ce premier TP de programmation fonctionnelle, nous reverrons les bases du langage abordées dans le premier cours. Les exercices devraient être faits dans l'ordre car leur difficulté est plus ou moins croissante. Prenez bien le temps de comprendre et de faire chaque exemple. Ces premiers exercices ne sont pas très amusants, mais ils constituent une base incontournable pour bien assimiler la syntaxe et l'inférence de type.

KOALA

Table des matières

Chapitre I

L'interprète

Vous devez utiliser l'interprète pour resoudre les exercices de ce TP. Je vous rappelle que la commande shell pour lancer l'interprète OCaml est `"ocaml"`.

Exemple :

```
1  >ocaml
2          Objective Caml version 3.10.2
3
4  #
```

Vous pouvez utiliser la commande rlwrap avec en paramètre l'interprète OCaml pour bénéficier d'une édition de ligne confortable.

La version d'OCaml que vous utilisez peut varier de celle de l'exemple, mais cela n'influera pas sur le contenu de ce module.
Pour quitter l'interprète, vous pouvez taper la directive `"#quit"` (avec le `#`) ou faire un `<ctrl> + d`.

Si vous voulez conserver le code que vous allez taper dans ce TP, vous pouvez l'écrire dans un fichier et le charger dans l'interprète grâce à la directive `"#use "fichier.ml""` (avec le `#` et les doubles quotes autour du nom du fichier de sources). Cette directive va lire, compiler et évaluer votre fichier directement dans l'interprète.

Le symbole ";;" sert à marquer la fin d'une commande dans l'interprète OCaml. Il est donc inutile de le mettre à la fin de vos expressions dans votre fichier de sources.

Exemple :

```
1   >cat example.ml
2   let greetings = "Bonjour les tech2 !" (* Notez l'absence de ";;" *)
3   >ocaml
4          Objective Caml version 3.11.1
5
6   # #use "example.ml";;
7   val greetings : string = "Bonjour les tech2 !"
8   # greetings;;
9   - : string = "Bonjour les tech2 !"
10  #
```

I.1 Exercice 1

- Créez un fichiez nommé `"exercice_1.ml"`
- Écrivez dans ce fichier `"let exercice_1 = "Réussi !""` suivi d'un retour à la ligne
- Sauvegardez, et, dans votre shell, lancez l'interprète OCaml
- À l'invite de l'interprète, utilisez la directive `"#use"` pour évaluer le fichier `"exercice_1.ml"`

Chapitre II

Expressions et types

Pour vous aider, je vous encourage très fortement à consulter la documentation officielle du langage à l'adresse http://caml.inria.fr/pub/docs/manual-ocaml/index.html et la documentation du module `Pervasives`. Ce module de la bibliothèque standard est ouvert par défaut dans tous vos programmes.

II.1 Exercice 2

Parcourez la documentation du module `Pervasives` à l'adresse http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html, principalement la section "comparaisons".

II.2 Exercice 3

Tentez de prédire le type et la valeur de chacune des expressions suivantes, puis vérifiez si vous avez raison à l'aide de l'interprète. Il est tout à fait possible que certaines de ces expressions s'évaluent en erreurs...

`En indentant les expressions, vous les rendrez plus lisibles donc plus faciles à typer.`

- let a = 42 ; ;
- a ; ;
- let b = "suspens..." ; ;
- let c = () ; ;
- let d = 42 + 0 ; ;
- let e = 42.0 +. 0 ; ;
- let f = 30 and g = 12 ; ;

- let h = f + g ; ;
- let i = let j = 50 and k = 8 in j - k ; ;
- let l = 42 in let m = l - 42 in l + m ; ;
- let n = 42 and o = n - 42 in n + o ; ;

II.3 Exercice 4

Tentez de prédire le type et la valeur de chacune des expressions suivantes, puis vérifiez si vous avez raison à l'aide de l'interprète. Il est tout à fait possible que certaines de ces expressions s'évaluent en erreurs...

- let fonction_p = fun a b -> a + b ; ;
- let fonction_q a b = a + b ; ;
- fonction_q 21 21 ; ;
- fonction_q ; ;
- let fonction_r () = 42 ; ;
- let fonction_s a = 42 ; ;
- let fonction_t a = a ; ;
- let fonction_u a b = a b ; ;
- let fonction_v a b c = a b c ; ;
- let fonction_w a b c = a (b c) ; ;
- let fonction_x () = let a = 42 in let b = 42 in a - b + 42 ; ;
- let y = "a" in let fonction_z a b = b ∧ y in fonction_z ; ;
- fonction_z ; ;

II.4 Exercice 5

Tentez de prédire le type et la valeur de chacune des expressions suivantes, puis vérifiez si vous avez raison à l'aide de l'interprète. Il est tout à fait possible que certaines de ces expressions s'évaluent en erreurs...

- let a = 42 in if a > 0 then true else false ; ;
- let str = "ocaml" in if str <> "" then print_endline str ; ;

- if 42 = 24 then (*) else (+) ; ;
- let _ = match 42 with 0 -> "zero" | n -> "42" ; ;
- 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 ; ;

Chapitre III

Fonctions

III.1 Exercice 6

Écrivez la fonction `incr` ayant pour type `int -> int` qui renvoie l'entier passé en paramètre incrémenté de 1.

Exemples :

```
# incr 0;;
- : int = 1
# incr 41;;
- : int = 42
```

III.2 Exercice 7

Écrivez la fonction factorielle ayant pour type `int -> int` défini de la façon suivante :

$$fact(n) = \begin{cases} n \times fact(n-1) & \text{si} \quad n \geq 1 \\ 1 & \text{si} \quad n = 0 \end{cases}$$

Votre fonction doit être récursive et utiliser une construction `if ... then ... else`. Voici un exemple de la sortie attendue :

```
# fact 0;;
- : int = 1
# fact 5;;
- : int = 120
# fact 7;;
- : int = 5040
```

III.3 Exercice 8

Ecrivez la fonction Fibonacci ayant pour type `int -> int` défini de la façon suivante :

$$fib(n) = \begin{cases} fib(n-1) + fib(n-2) & \text{si} \quad n \geq 2 \\ 1 & \text{si} \quad n = 1 \\ 0 & \text{si} \quad n = 0 \end{cases}$$

Votre fonction doit être récursive et **ne doit pas** utiliser une construction `if ... then ... else`. À la place, vous devez utiliser un filtrage. Voici un exemple de la sortie attendue :

```
# fib 0;;
- : int = 0
# fib 1;;
- : int = 1
# fib 2;;
- : int = 1
# fib 13;;
- : int = 233
# fib 25;;
- : int = 75025
```

Chapitre IV

Conclusion

Nous éspérons que vous avez apprécié ce sujet autant que nous avons apprécié le rédiger pour vous.

Vos avis sont très importants pour nous et nous permettent chaque jour d'améliorer nos contenus. C'est pourquoi nous comptons beaucoup sur vous pour nous apporter vos retours.

Si vous trouvez que certains points du sujet sont obscurs, pas assez bien expliqués ou tout simplement contiennent des fautes d'orthographe, signalez-le nous. Pour cela, il vous suffit de nous envoyer un mail à l'adresse koala@epitech.eu.

Pour aller plus loin dans votre apprentissage de la programmation fonctionnelle, nous vous conseillons le livre "Développement d'applications avec Objective CAML" : http://bibliotech.epitech.eu/?page=book&id=156 ou sa version en anglais dont le PDF est disponible : http://bibliotech.epitech.eu/?page=book&id=155.
Il existe bien sûr d'autres livres sur la programmation fonctionnelle : http://bibliotech.epitech.eu/?page=search&categ=15