Programmation fonctionnelle

Fiche de TP 1

L3 Informatique 2020-2021

Premiers pas

Exercice 1. (Au toplevel)

Le but de cet exercice est de se familiariser avec l'interpréteur `ocaml` et les types de base. Nous allons entrer des **expressions** que l'interpréteur évaluera (quand elles se terminent par `;;`).

1. Lancer le toplevel et évaluer l'entier `33`. Recommencer. Pouvez-vous utiliser les flèches directionnelles ?
2. Quitter le toplevel et le relancer avec la commande `rlwrap`. Refaire la question précédente. Quelles sont les informations que vous donne le toplevel ?
3. Déterminer les types des expressions suivantes (lorsque ce sont des expressions CAML). Vérifier en utilisant le toplevel et commenter.

   (a) `2`
   (b) `2.0`
   (c) `2,0`
   (d) `2;0`
   (e) `(2, 0)`
   (f) `(2; 0)`
   (g) `a`
   (h) `'a'`
   (i) `"a"`
   (j) `true`
   (k) `()`
   (l) `[]`
   (m) `[1]`
   (n) `[1, true]`
   (o) `[1; true]`

   Les types de base de CAML sont consultables à l'adresse

   https://caml.inria.fr/pub/docs/manual-ocaml/coreexamples.html#sec9.

4. Lorsque c'est possible, donner des exemples d'expressions ayant les types suivants. Commenter.

(a) `int * float`

(b) `(int, float)`

(c) `string list`

(d) `bool list * string`

(e) `'a * int`

(f) `int list list`

5. Que valent les calculs suivants ? Après avoir répondu, tester au toplevel et expliquer.

(a) `1 + 2`

(b) `1.1 + 2.2`

(c) `1.1 + 2`

(d) `2 / 3`

(e) `7 mod 2`

(f) `7. mod 3.`

(g) `int_of_float (2. ** 3.)`

(h) `2 = 3`

(i) `'a' = 'b'`

(j) `"a" = 'a'`

(k) `not 1 = 0`

(l) `not (1 = 0)`

6. Utiliser le toplevel pour tester si les opérateurs booléens *et* (`&&`) et *ou* (`||`) sont séquentiels (c'est à dire que la partie gauche est évaluée d'abord et la partie droite seulement si nécessaire). Donner les tests réalisés pour s'en assurer. Il est possible pour cela d'utiliser l'instruction `print_string` qui affiche la chaîne de caractères en argument.

Exercice 2. (Liaison locale)

Dans le paradigme fonctionnel, la notion de variable (et donc d'affectation) n'existe pas. Pour nommer des expressions, on utilise le mécanisme de *liaison*.

Pour réaliser une *liaison locale*, on utilise la syntaxe suivante :

```
let nom = expr1 in expr2;;
```

Ceci est une expression et possède donc une valeur : c'est la valeur de `expr2` dans laquelle toutes les occurrences (libres) de `nom` sont remplacées par `expr1`.

Le mot-clé `and` permet de faire des liaisons simultanées :

```
let nom1 = expr1 and nom2 = expr2 in expr3;;
```

1. Considérons un cylindre de hauteur h et un rayon r. Commencer par calculer l'aire d du disque qui forme la base du cylindre (πr^2) en une seule expression et en utilisant des liaisons locales pour π, r et h.

   Il est possible d'obtenir la valeur de π en utilisant la fonction arc cosinus (`let pi = acos (-1.) in ...`).

2. Reprendre l'expression précédente pour calculer, toujours en une seule expression, le triplet (p, a, v) où p est le périmètre de la base ($2\pi r$), a est l'aire du cylindre ($2d + p \times h$) et v son volume ($d \times h$). Bien entendu, les expressions apparaissant dans plusieurs calculs différents ne doivent être calculées qu'une seule fois.

Exercice 3. (Différentes liaisons)

Il existe un deuxième mécanisme de liaison — la *liaison globale* — permettant notamment de définir des constantes et les fonctions. **Attention, une liaison locale est une *expression* alors qu'un liaison globale ne l'est pas.** Une liaison globale ne réalise pas non plus un effet de bord : sa raison d'être est de lier un nom à une valeur. Grâce au principe de transparence référentielle, toutes les occurrences du nom considéré peuvent se remplacer dans les expressions futures sans changer leur comportement.

Pour réaliser une *liaison globale*, on utilise la syntaxe suivante :

```
let nom = expr;;
```

Comme précédemment, le mot-clé `and` permet de faire des liaisons simultanées :

```
let nom1 = expr1 and nom2 = expr2;;
```

1. Peut-on écrire les morceaux de code suivants ? **Réfléchir avant de tester !**

   (a)
   ```
   let a = 1 in a + 2;;
   a + 3;;
   ```

   (b)
   ```
   let b = 5;;
   let b = 5.5;;
   ```

   (c)
   ```
   let c = 1;;
   let d = c;;
   c + d;;
   ```

   (d)
   ```
   let e = 1 let f = e;;
   ```

   (e)
   ```
   let g = 1 and h = g;;
   ```

   (f)
   ```
   let a = 1 in let b = a;;
   ```

   (g)
   ```
   let a = 1 in let b = a in b;;
   ```

2. Qu'affiche le code suivant ? **Réfléchir avant de tester !**

```
let a = 1;;
let a = 1.2 in a;;
a;;

let a = 1 in
    let a = 2 and b = a in
        a + b;;
```

Exercice 4. (Alternative)

La structure de contrôle conditionnelle (ou alternative) en CAML possède la syntaxe

```
if expr1 then expr2 else expr3
```

Attention, il s'agit d'une expression, elle a donc un type et une valeur (c'est l'équivalent du `if` ternaire du C). L'expression `expr1` est de type `bool` et les expressions `expr2` et `expr3` doivent être du même type (qui est également le type de l'expression complète).

1. Calculer le maximum de `a` et `b`.
2. Calculer le minimum de `a`, `b` et `c`.
3. Que penser du code suivant ? **Réfléchir avant de tester !**

   ```
   if a mod 2 = 0 then a else "odd";;
   ```

4. Qu'est-ce qui ne va pas dans le code suivant ? **Réfléchir avant de tester !**

   ```
   if a < 10 then let b = "small" else let b = "large";;
   ```

   Écrire le code correct qui permet de de lier le nom `b` à l'une des chaînes `"small"` ou `"large"` en fonction de la valeur de `a`.
5. Étant donné un entier `a` et en utilisant une conditionnelle, donner le code permettant d'obtenir $b = \lceil a/2 \rceil$.
6. Étant donnés trois entiers `a`, `b` et `c`, calculer l'expression suivante sans faire deux fois le même test ou le même calcul :

$$\begin{cases} \min(a,b)^2 + 1 & \text{si } c \text{ est divisible par 3,} \\ \min(a,b)^2 & \text{sinon.} \end{cases}$$

Exercice 5. (Fonctions)

En CAML, on dispose d'un certain nombre de librairies prédéfinies (modules). Le module `Pervasives` est chargé par défaut et sa documentation se trouve à l'adresse

http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html

et les autres librairies standards à l'adresse

http://caml.inria.fr/pub/docs/manual-ocaml/stdlib.html.

Dans cet exercice, nous allons définir nos premières fonctions. Celles-ci sont à écrire dans un fichier `CAML_tp1.ml`. Le contenu de ce fichier se charge dans le toplevel par la commande (attention à ne pas oublier le « # ») :

```
# #use "CAML_tp1.ml";;
```

Pour *appliquer* une fonction, il suffit de donner son expression suivie des arguments **sans parenthèses**, contrairement aux autres langages usuels. Par exemple,

```
int_of_float 3.4;;
mod_float 3.4 2.0;;
(+) 1 2;;
String.make 5 'A';;
(fun x -> x + 1) 4;;
```

Il est possible de parenthéser globalement une application de fonction lors d'applications imbriquées. Par exemple,

```
max 1 (max 2 3);;
```

Enfin pour définir ses propres fonctions, on peut utiliser (entre autres) la syntaxe

```
let nom p1 ... pn = expr (* 'expr' est le corps de la fonction. *)
```

1. Écrire et tester une fonction `average` qui calcule la moyenne de trois entiers. Cette fonction peut-elle être utilisée avec des flottants ?
2. Écrire une fonction `implies` qui prend deux expressions booléennes `a` et `b` et renvoie vrai si `a` implique `b` au sens logique.
3. Écrire une fonction `inv` qui prend un couple et renvoie le couple inversé. Faire deux versions : l'une en utilisant `fst` et `snd` et l'autre sans.
4. Écrire la même fonction, mais uniquement pour un couple d'entiers. Vérifier son type.
5. Écrire la fonction sans argument `f_one` qui renvoie la constante 1. Vérifier qu'elle a bien le type attendu.
6. Qu'affiche le code suivant (pour chaque ligne) ? **Réfléchir avant de tester !**

```
let m = 3;;
let f x = x;;
let g x = x + m;;
f 4;;
g 4;;
let m = 5;;
g 4;;
f m;;
```