Institut National Universitaire Champollion

# TP 9 - Un petit évaluateur Caml

**Programmation fonctionnelle en Caml**
**L3 INFO - Semestre 5**

## Fonctionnelles

1. Déterminer le type de la fonction suivante et tester :

   ```
   let fzz = fun f a -> f(a+1) = 0  ; ;
   ```

2. Pour chaque requête suivante, donner la réplique Caml et commenter

   ```
   fzz(3)  ; ;
   let g = fzz(fun n -> 3)  ; ;
   g 4  ; ;
   ```

3. Construire un exemple de fonction de chaque type suivant :

   (a) `int -> bool -> float`

   (b) `(int -> bool) -> float`

## Un petit évaluateur Caml

Nous avons étudié les modes d'évaluation des différentes expressions Caml. Le but de cet exercice est d'écrire un petit interpréteur Caml reconnaissant quelques expressions arithmétiques en nombre entier et les définitions globales et locales. Nous proposons d'écrire un évaluateur pour cette version très allégée de Caml que nous appellerons *Caml O'Chocolat*.

### Les expressions Caml O'Chocolat

En Caml O'Chocolat, une expression peut être une constante entière, un identificateur (réduit à un seul caractère), la somme de deux expressions, le produit de deux expressions ou une expression à une certaine puissance entière. Nous définissons donc le type expression comme suit :

```
type Expression = Const of int | Var of char
| Add of Expression*Expression
| Mult of Expression*Expression
| Puiss of Expression*int ; ;
```

1. Définir deux expressions $e_1$ et $e_2$ représentant respectivement $1+2x^3$ et $1+a^2$.

   Pour évaluer nos expressions, nous devons disposer d'un environnement courant. On définit une liaison entre un identificateur et la valeur d'une expression par le type produit suivant :

   ```
   type liaison = {id   : char ; valeur   : int} ; ;
   ```

On peut alors définir un environnement comme une liste de liaisons. On pourra supposer qu'initialement, notre environnement courant, que nous noterons envC est :

```
let envC = [{id='a' ;valeur=3} ;{id='b' ;valeur=4}] ; ;
```

2. Écrire une fonction `evalVar : char * liaison list -> int` qui, à un caractère *c* et à un environnement *env* associe la valeur entière liée à *c* dans l'environnement *env* si cette liaison existe et un message d'erreur sinon.

```
evalVar ('a',envC) ; ;
- : int = 3
evalVar ('b',envC) ; ;
- : int = 4
evalVar ('x',envC) ; ;
Exception non rattrapée : Failure " identificateur inconnu"
```

3. Écrire une fonction `puissance : int * int -> int` qui calcule (naïvement) la puissance entière puissance entière positive d'un nombre entier. On affichera une erreur si la puissance est strictement négative.
4. Écrire une fonctionnelle curryfiée `evalExp :liaison list -> Expression -> int` qui, à un environnement et une expression donnés associe la valeur de cette expression dans l'environnement ou un message d'erreur si l'expression contient un identificateur non lié.
   *On définira une fonction récursive auxiliaire locale ne portant que sur l'expression à évaluer et on pourra bien sûr utiliser `evalVar` et puissance.*

## Définitions globales et locales :

Une définition crée la liaison entre un identificateur et la valeur d'une expression. Nous représentons une définition par le type produit suivant :

```
type Definition={ident :char ; exp : Expression } ; ;
```

1. Écrire une fonction `ajoute : liaison list -> Definition -> liaison list` qui, à un environnement *env* et à une définition *d* associe le nouvel environnement obtenu en ajoutant à *env* la liaison entre l'identificateur et la valeur de l'expression dans l'environnement *env*.

```
ajoute envC {ident='x' ;exp = Add(Var 'a',Const 3)} ; ;
- : liaison list =
[{id = 'x' ; valeur = 6} ; {id = 'a' ; valeur = 3} ; {id = 'b' ; valeur = 4}]
```

## Programmes Caml O'Chocolat

Un programme Caml O'Chocolat peut être soit une expression, soit une définition globale, soit une définition locale permettant la réalisation d'une définition le temps de l'évaluation d'un programme. On définit donc le type récursif suivant :

```
type Programme =
Elementaire of Expression
| DefGlob of Definition
| DefLocale of Definition*Programme ; ;
```

Le petit programme Caml O'Chocolat suivant :

```
Soit x=7 dans
    Soit y = x + 3 dans x + 3*y  ; ;
```

sera alors représenté par l'objet de type `Programme` suivant :

```
let p  = DefLocale ({ident = ‘x‘ ; exp = Const 7},
DefLocale ({ident = ‘y‘ ; exp = Add (Var ‘x‘, Const 3)},
Elementaire (Add (Var ‘x‘, Mult (Const 3, Var ‘y‘))))) ; ;
```

1. Écrire une fonction `evalProg : Programme * liaison list -> int * liaison list` qui, à un programme et un environnement *env* donnés, associe la valeur de ce programme dans l'environnement *env* (la valleur affichée par la réplique Caml) et le nouvel environnement.

```
Pour le programme  : 1+a2  ; ;
#evalProg( Elementaire e2,envC) ; ;
-  : int * liaison list = 10, [{id = ’a’ ; valeur = 3} ; {id = ’b’ ; valeur = 4}]


Pour le programme  : soit x = 1+a2  ; ;
#evalProg( DefGlob {ident=’x’  ; exp=e2 },envC) ; ;
-  : int * liaison list =
10, [{id = ’x’ ; valeur = 10} ; {id = ’a’ ; valeur = 3} ; {id = ’b’ ; valeur = 4}]


Pour le programme  : soit x = 7 dans soit y = x+3 dans x+3*y  ; ;
#evalProg ( p ,envC) ; ;
-  : int * liaison list = 37, [{id = ’a’ ; valeur = 3} ; {id = ’b’ ; valeur = 4}]
```