▶ **HX4 & HX1 – Option Informatique**
Année 2011, Cinquième TP Caml
Louis Jachiet (http://www.eleves.ens.fr/home/jachiet/)

MaCalc v42

Aujourd'hui on va fabriquer une petite calculatrice, MaCalc v42. Elle ressemble beaucoup à celles que vous utilisiez au lycée (et que vous utilisez peut être même encore).

1 Représentation sous forme d'arbre

Les expressions mathématiques que l'on va manipuler seront sous la forme d'arbres d'expressions, avec des variables, des constantes et des opérateurs (+,-,*,/). Par exemple, pour représenter $5/x*9+7$ on obtient l'arbre :

```
          +
         / \
       *     7
      / \
     /    9
    / \
   5   x
```

1.1 Le type

Vous devez utiliser le type :

```
type Expr =
  | Plus of Expr*Expr
  | Moins of Expr*Expr
  | Fois of Expr*Expr
  | Div of Expr*Expr
  | Const of float
  | Var of string
```

1.2 Manipulations simples

▶ **Question 1** *Écrire une fonction qui prend en argument une Expr et retourne la valeur représentée. On peut déclencher une exception quand une variable apparait dans l'expression*

▶ **Question 2** *Écrire une fonction qui prend en argument une string et une Expr et retourne une Expr correspondant à la dérivée de la fonction par rapport à la variable désignée par la string. On ne cherche pas à avoir l'expression la plus simple possible mais juste à avoir une expression correcte.*

1.3 Variables

Pour manipuler les variables, on va utiliser ce que l'on appelle un environnement. Un environnement c'est un objet qui donne la valeur que l'on attribue à chaque variable. Une manière simple de représenter l'environnement c'est d'avoir une $(string, float)$ *list*. Quand $("x", 3.14)$ apparait dans la liste, la liste ne contenant pas de doublon, c'est que x vaut 3.14. Par chance, Caml-light dispose d'une fonction

```
assoc : : 'a -> ('a * 'b) list -> 'b
```

En appelant *assoc x liste*, on obtient, si elle existe, la valeur associée à x.

▶ **Question 3** ***Évaluation avec des variables*** *Reprendre votre solution à la question 1 en gérant les variables. Votre fonction doit maintenant prendre en paramètre un environnement et une Expr.*

2 Tracer des expressions

Votre calculatrice de lycée vous permettait de tracer des courbes ? MaCalc peut aussi le faire !

```
#open "graphics";; (*a faire une fois *)
open_graph "" (* pour ouvrir une fenetre *)
set_color (rgb 255 0 0 ) ; (* change la couleur du pinceau *)
moveto x y ; (* deplace le pinceau *)
lineto x y ; (* deplace le pinceau en dessinant *)
```

▶ **Question 4** *Écrire un programme qui prend en argument e une Expr, x une string, a, b, c, d des $float$ et qui trace $e(x), x \in [a, b]$, pour une fenêtre qui a un intervalle de $y = [c, d]$.*

3 Lecture des expressions

Dans votre calculatrice de lycée, vous ne tapiez pas $Plus(Fois(Const 2., Var "x"), Const 5.)$, non ? On s'intéresse donc à la manière de récupérer les expressions de manière naturelle. Tout d'abord, il nous faut une fonction qui lit des entiers dans une chaine.

3.1 La notation polonaise

La notation polonaise est une manière d'écrire les calculs sans avoir besoin de mettre de parenthèse. Tout en étant relativement simple pour les humains, elle est surtout très simple à lire par ordinateur. La notation polonaise est une notation préfixée, c'est à dire qu'au lieu d'écrire $14 + 28$ on écrit + 14 28. Au lieu d'écrire $(3 + 4) * 6$ on écrit * + 3 4 6. Elle peut sembler peut naturelle pour un humain mais en fait c'est celle qu'on emploie quand on lit le calcul (enfin presque). Pour $(3 + 4) * 6$ on lit *Produit de la somme de 3 et 4 avec 6.*

▶ **Question 5** *Transformer de notation polonaise à notation 'habituelle' ou l'inverse les expressions suivantes :*
– *+ * - 3 5 + 5 6 8*
– *+ 2 + 6 + 5 7*
– *6*47+451+57*45*4+5*8/42*

▶ **Question 6** *Écrire une fonction lit_int qui lit un entier dans une chaine. C'est à dire, étant donné une chaîne s, un entier i renvoie (v, j) avec j maximal tel que les caractères i à $j - 1$ correspondent à l'entier v. Pour savoir un caractère c correspond à un entier on peut utiliser la condition suivante :* $c \geq$ '0'&&$c \leq$ '9', *pour récupérer la valeur d'un chiffre on peut utiliser :* $int\_of\_char\ c - int\_of\_char$ '0'

▶ **Question 7** *Écrire une fonction lit_var qui fonctionne de la même manière que lit_int. On peut, de même, utiliser* $c \geq$ 'a'&&$c \leq$ 'z' *et string_of_char.*

▶ **Question 8** *En déduire une fonction lit_expr qui lit une expression en notation polonaise dans une fonction.*

▶ **Question 9** *Compiler les 3 fonctions précédentes en une fonction parse qui prend une chaîne de caractère et renvoie l'expression correspondante.*

▶ **Question 10** *Écrire une fonction simplifie qui prend en argument une expression qui renvoie une expression simplifiée. On peut, par exemple, supprimer les multiplications par 1 ou 0, pré-calculer certaines branches, simplifier ...*

4 Pour ceux qui s'ennuient

▶ **Question 11** *Écrire la table des inverses des nombres de* $\mathbb{Z}/17\mathbb{Z}$.

▶ **Question 12** *Écrire une quine, c'est à dire un programme qui affiche son propre code source.*

■

MaCalc v42

Un corrigé

▶ Question 1

```
let rec eval = function
  | Plus(a,b) -> eval a +. eval b
  | Moins(a,b) -> eval a -. eval b
  | Fois(a,b) -> eval a *. eval b
  | Div(a,b) -> eval a /. eval b
  | Const f -> f
  | Var a -> failwith "variable"
```

▶ Question 2

```
let rec diff x = function
  | Plus(a,b) -> Plus(diff x a,diff x b)
  | Moins(a,b) ->Moins(diff x a,diff x b)
  | Fois(a,b) -> Plus(Fois(diff x a,b),Fois(diff x b,a))
  | Div(a,b) -> Div(Moins(Fois(diff x a,b),Fois(diff x b,a)),Fois(a,b))
  | Const(f) -> Const(0.)
  | Var a -> if a = x then Const(1.) else Const(0.)
```

▶ Question 3

```
let rec eval2 env = function
  | Plus(a,b) -> eval2 env a +. eval2 env b
  | Moins(a,b) -> eval2 env a -. eval2 env b
  | Fois(a,b) -> eval2 env a *. eval2 env b
  | Div(a,b) -> eval2 env a /. eval2 env b
  | Const(f) -> f
  | Var a -> assoc a env
```

▶ Question 4

```
let trace e v a b c d =
  open_graph " 800x800" ;
  set_color (rgb 0 0 255) ;
  moveto 0 400 ;
  lineto 800 400 ;
  moveto 400 0 ;
  lineto 400 800 ;
  set_color (rgb 255 0 0) ;
  moveto 0 (int_of_float (800./.(d-.c)*.(eval2 [v,a] e -. c))) ;
  for x = 1 to 800 do
    lineto x (int_of_float (800./.(d-.c)*.(eval2 [v,((float_of_int x)*.(b-.a)/.800.+.a)] e -. c)));
  done
```

▶ Question 9

```
let parse spre =
  let s = spre ^ " " in
  let rec lit_int v i =
    if s.[i]>='0' && s.[i]<='9'
    then lit_int (v*10+int_of_char s.[i]-int_of_char '0') (i+1)
    else (Const (float_of_int v),i)
  in
  let rec lit_var deb i =
    if s.[i]>='a' && s.[i]<='z'
    then lit_var (deb^(string_of_char s.[i])) (i+1)
    else (Var deb,i)
  in
  let rec lit_expr i =
    let (v,ni) = lit_var "" i in
    if ni<>i
    then (v,ni)
    else
      let (v,ni) = lit_int 0 i in
      if ni <> i
      then (v,ni)
      else
        let (e1,i1) = lit_expr (i+2) in
        let (e2,i2) = lit_expr (i1+1) in
        (match s.[i] with
          | '+' -> Plus(e1,e2),i2
          | '-' -> Moins(e1,e2),i2
          | '*' -> Fois(e1,e2),i2
          | _ -> Div (e1,e2),i2)
  in
  fst (lit_expr 0)
```

▶ Question 11

```
let divi p =
  let rec exp a = function
    | 0 -> 1
    | n ->
      let s = exp a (n/2) in
      if n mod 2 = 0
      then
        (s*s) mod p
      else
        (s*s*a) mod p
  in
  for y = 1 to p-1 do
    print_int (exp y (p-2)) ; print_string " "
  done

let a = divi 17
```

▶ Question 12

```
(fun s -> print_string s ; print_char (char_of_int 34) ;
print_string s ; print_char (char_of_int 34) )
"(fun s -> print_string s ; print_char (char_of_int 34) ;
 print_string s ; print_char (char_of_int 34) )"
```