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

Premiers pas en Caml

Vous avez déjà quelques notions de Caml, il vous faut maintenant passer à la pratique. Et pour cela choisir et maîtriser une interface. Une bonne solution, quelque soit le système d'exploitation, c'est Emacs avec le tuareg mode. Cependant, pour ceux qui ne connaissent pas déjà Emacs et qui veulent aller à la simplicité, il est plus facile de programmer en choisissant une autre interface. Les utilisateurs de mac peuvent s'orienter vers camlx et ceux de windows vers le notepad avec l'interface par défaut de caml sous windows. À tout moment, n'hésitez pas à me poser des questions.

1 Préliminaires

Pour vous familiariser avec l'interface, vous allez commencer par des programmes simples. Les solutions aux questions suivantes tiennent en une ligne.

1.1 Calculs

▶ **Question 1** *Faites faire à Caml les calculs suivants :* $1+1$, $2*3$, *"bonjour" concaténé avec " le monde !". (Les guillemets ne sont là que pour délimiter les chaînes.) Comparez pour l'ordre* $<$ *de caml : "hx1","hx4", "HX4" et "HX1", true avec false.*

▶ **Question 2** *Corrigez les expressions suivantes pour qu'elles aient un sens et soient correctes et effectuez les :* 2.3^{42}, $6.4 * 6$, *"24"+"18".*

1.2 Affichage

Pour l'affichage on utilise les fonctions :

```
print_type : type -> unit
```

où type est *int, string, float, char*

▶ **Question 3** ***Hello world !***
Faites un programme qui affiche : Hello world !

▶ **Question 4** *Faites un programme qui affiche le résultat des expressions suivantes :* $2 + 3$, $2.0^{42.0}$.

2 Variables et fonctions

2.1 Variables

En Caml on peut définir des variables dites *globales* de la manière suivante :

```
let x = expr
```

ou *locales* de la manière suivante :

```
let x = expr1 in expr2
```

mais de cette seconde manière x ne vaudra $expr1$ que dans $expr2$. Après il vaudra ce qu'il valait avant et donc rien s'il ne valait rien avant. La première expression n'a ni type ni valeur tandis que la seconde a le type et la valeur de $expr2$.
Par exemple :

```
let x = 3 ;;
let y = 2 ;;
print_int (x*x*y) ;;
```

affiche 18 et

```
let x = 42 in
  (let y = x*x in
    y+1)
;;
```

nous renvoie - *: int = 1765*

▶ **Question 5 *** *Sans recopier le code dans caml, devinez-vous ce que renvoie :*

```
let x = 7 in
  let x = 2 in
    (let x = x*x in
      x+1)+x
;;
```

?

▶ **Question 6** *Faites le calcul suivant en utilisant des variables pour ne pas recopier les termes qui apparaissent plusieurs fois et simplifier la lecture (mais sans simplifier l'expression) :*

$$\frac{\frac{(23*34)^2}{8} + 6*7*23*34*9}{3} + (6*7)^2$$

2.2 fonctions

En Caml, il n'y a pas de vraie différence entre une fonction (ce qui est déclaré par un *proc* en Maple) et un type de base *(int, string, ...)*. C'est donc naturellement que l'on déclare les fonctions avec des **let**. La définition d'une fonction se fait de la façon suivante :

```
let nom_fonction arg1 arg2 ... argn = expr
```

ou, si l'on veut faire une fonction locale :

```
let nom_fonction arg1 arg2 ... argn = expr in expr2
```

Ainsi si l'on veut faire une fonction qui donne le carré d'un entier on écrit

```
let carre x = x*x ;;
```

▶ **Question 7** *Écrivez une fonction qui calcule le produit de deux nombres entiers, puis de deux nombres réels.*

▶ **Question 8** *Écrivez une fonction f qui, étant donné des $float$ x et y, renvoie $\dfrac{y/x + x}{2}$.*

Note En caml, il existe la structure **if**, **then**, **else** qu'on utilise de la facon suivante :

```
if condition
then expr1
else expr2
```

(la même qu'en Maple) et où $expr1$ et $expr2$ doivent avoir le même type qui sera le type de toute l'expression. *condition* doit être un booléen.

▶ **Question 9** *Écrivez une fonction fabs qui renvoie la valeur absolue d'un $float$.*

▶ **Question 10** *Programmez la fonction*

```
print_bool : bool -> unit
```

qui affiche "True" ou "False" selon la valeur du booléen donné en paramètre. Affichez le résultat du calcul $20.0^{30.0} > 30.0^{20.0}$.

2.3 le mot clé **rec**

Il arrive parfois que l'on veuille définir quelque chose en fonction de lui même par exemple une suite récursive ($u_{n+1} = f(u_n)$). Aussi caml permet de le faire avec l'aide du mot clé **rec**. Pour cela on rajoute **rec** juste après **let** dans un définition. Par exemple, la factorielle est définie comme étant 1 pour 0! et $n*((n-1)!)$ sinon. On écrit donc en caml :

```
let rec factorielle n =
 if n = 0
 then 1
 else n*(factorielle (n-1)) ;;
```

▶ **Question 11** *Faites une fonction qui prend en argument un entier n et un float y et qui donne le n^e terme de la suite récursive définie à partir de la fonction $x \mapsto f(x, y)$ de la question 8 et d'un $u_0 = 1$. Vers quoi converge cette suite à y fixé ?*

▶ **Question 12** *Écrivez une fonction récursive qui prend en argument deux entiers : $p, n \geq 1$ et qui renvoie le plus grand entier $k \leq n$ tel que $p \bmod k = 0$. En déduire une fonction*

```
est_premier : int -> bool
```

qui teste la primalité de son argument.

▶ **Question 13** *Écrivez un programme qui étant donné une fonction continue f et deux abscisses a et b tels que $f(a) \leq 0$, $f(b) \geq 0$ et ε (tous des $float$), revoie x tel que $|f(x)| \leq \varepsilon$. La précision du float peut vous poser un problème pour certains couples f, ε mais ce n'est pas grave car votre fonction ne sera utilisée que dans des cas raisonnables. (Cependant, l'élégance voudrait que vous programmiez une fonction qui termine, quitte à s'autoriser un petit écart à la condition $|f(x)| \leq \varepsilon$.)*

▶ **Question 14 *** *Écrivez une fonction*

```
racine_nieme : int -> float -> float
```

qui, étant donné n et x, renvoie $\sqrt[n]{x}$. Vous ne devez pas utiliser de conversions int vers $float$ (ou l'inverse) dans cette question. Pensez à ré-utiliser les questions précédentes.

▶ **Question 15 *** *Écrivez les fonctions de conversions des $float$ vers les int et réciproquement. Si votre solution est lente (si votre fonction tourne plus d'un millième de seconde) sur les entrées 1e09 ou 1000 000 000, recodez-les plus de manière plus rapide.*

Note : les fonctions de la bibliothèque standard sont de la forme

```
type1_of_type2
```

mais utilisent quand c'est possible des instructions processeurs elles sont donc quasi-instantanées. ■

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

Premiers pas en Caml

Un corrigé

▶ Question 1

```
1+1;;2*3;;
"bonjour"^ "␣le␣monde!";;
"hx4">"hx1";;"hx1">"HX4";;"HX4">"HX1";;
false<true;;
```

▶ Question 2

```
2.3**42.0;;
2*3;;
24+18;; ou "24"^"18";;
```

▶ Question 3

```
print_string "Hello␣world!";;
```

▶ Question 4

```
print_int (2+3);;
print_float (2.0**42.0);;
```

▶ Question 5 $(4+1)+2$ donc 7

▶ Question 6

```
let a = 23*34 in
  let b = 6*7 in
    let num = a*a/8+b*a*9 in
      num/3+b*b
```

▶ Question 7

```
let int_prod x y = x*y ;;
let float_prod x y = x*.y ;;
```

▶ Question 8

```
let f x y = ((y/.x)+.x)/.2.0 ;;
```

▶ Question 9

```
let fabs a =
  if a < 0.
  then -.a
  else a
;;
```

▶ Question 10

```
let print_bool b = print_string (if b then "True\n" else "False\n");;
print_bool (20.0**30.0>30.0**20.0);;
```

▶ Question 11 Cette suite converge vers $\sqrt{y}$

```
let rec terme y n =
  if n = 0
  then 1.
  else f(terme y (n-1))
;;
```

▶ Question 12

```
let rec diviseur p n =
  if p mod n = 0
  then n
  else diviseur p (n-1)
;;

let est_premier p =
  p>1 && (diviseur p (p-1) = 1)
;;
```

▶ Question 13

```
let rec dichotomie f a b epsilon =
  let mil = (a+.b)/.2. in
  if fabs (f mil) < epsilon || a = mil || b = mil
  then mil
  else
    if (f mil) < 0.0
    then dichotomie f mil b epsilon
    else dichotomie f a mil epsilon
;;
```

▶ Question 14

```
let racine_nieme n x =
  let poly y =
   let rec foo n =
     if n = 0
     then 1.
     else y*.(foo (n-1))
   in
   (foo n)-.x in
  dichotomie poly 0. (x+.1.) 0.
;;
```

▶ Question 15

```
let rec itof i =
  if i < 0
  then (-1.0)*.itof (-i)
  else
    if i = 0
    then 0.0
    else
      (if i mod 2 = 1
       then 1.0
      else 0.0)+.2.*.(itof (i/2))
;;

let ftoi f =
  let rec foo f =
    if f < 0.5
    then (0,f)
    else
      let (ss,sf) = foo (f/.2.) in
      if sf<0.5
      then (ss*2,sf*.2.)
      else (ss*2+1,(sf-.0.5)*.2.)
  in
  (if f < 0.0 then (-1) else 1)*(fst (foo (fabs f)))
;;
```