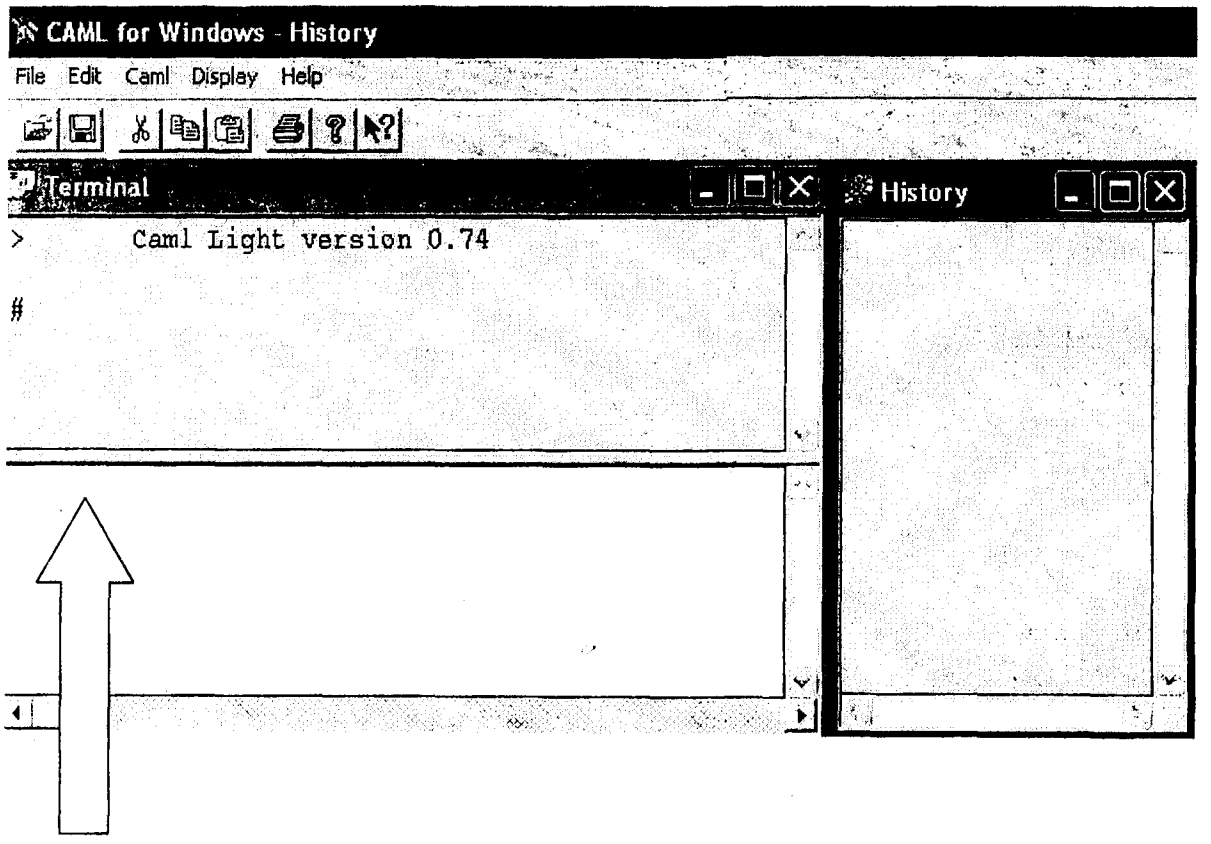
1. Lancement de CaML light

Ceci a pour effet de déclencher l'exécution de la boucle de **Lecture/Evaluation/Impression** qui se manifeste par les 2 fenêtres : Terminal (Editeur et Interprétation) et History (trace des expressions).



EDITEUR

En mode interactif, CaML donne systématiquement une réponse qui contient :

- Le nom de la variable déclarée s'il y en a.
- Le type trouvé pendant le typage.
- La valeur calculée après exécution.

2. Evaluation d'expressions

Pour vous entrainer à l'utilisation de CAML, demandez à l'interprète d'évaluer les expressions suivantes (et chercher à chaque fois à donner un sens à la réponse de l'interprète) :

Expressions sur les entiers :

1098 ;;	5 * 99 ;;
10 / 5 ;;	10 / 6 ;;
276 mod 23 ;;	(3 * 5) + (10 – 6) ;;
21 + 35 + 12 + 7 ;;	2.7 + 10 ;;
int_of_float 1.;;	int_of_float 1.1;;

Expressions sur les réels (flottants) :

3.45;;	10.0 /. 6.0 ;;
1.1 +. 2.2 ;;	ceil 3.14 ;;
floor 3.14 ;;	exp 1.0 ;;
log 2.71828182846 ;;	3. ** 2. ;;
3.45e10;;	sqrt 25. ;;

Expressions sur les booléens :

true ;;	not true ;;
true && false ;;	true or false ;;
1<2;;	1<=2;;
0<1 & 3<9;;	float_of_int 76;;
" un" < " deux"	(not false) or true ;;

Expressions sur les caractères et chaînes de caractères :

`&` ;;	`a` ;;
int_of_char `a`;; (donne un type entier)	char_of_int 97;;
"a";;	"Ceci est une ch" ^ "aine de caractères !"
"toutou".[5] ;;	"kArim".[1] ;;
string_length "KADER" ;;	sub_string "bonjour" 0 3

Expressions sur les Uplets

(1,2) ;;	("coucou",3.1,('A',2)) ;;
1,2,3 ;;	fst (12, "octobre") ;;
(1,(2,3)) ;;	snd (12, "octobre") ;;
(12, "octobre") ;;	
(12, "octobre",true) ;;	

TP 1 CaML Light

Définition :
Une définition est une déclaration qui associe un nom à une valeur. On distingue les déclarations globales les déclarations locales.

Expressions de déclarations globales :
Syntaxe : **let** *nom* = *expr* **;;**
où *nom* représente l'identificateur et *expr* l'expression qui lui est associée.

let x = 5 ;;	let pi = 3.14159 ;;
x ;;	let rayon = 10.0 ;;
x + 2 ;;	(pi *. rayon *. rayon) ;;
	let circonference = (2.0 *. pi *. rayon) ;;
let taille = 2.0 ;;	
taille ;;	
5 * taille ;;	

Expressions de déclarations globales simultanées :
Syntaxe : **let** $nom_1 = expr_1$
and $nom_2 = expr_2$
.
.
and $nom_n = expr_n$ **;;**

let a= 3 and b= 2 ;;	
a + b ;;	

Expressions de déclarations locales :
Syntaxe : **let** *nom* = $expr_1$ **in** $expr_2$**;;**
la valeur $expr_1$ du nom *nom*, n'est connue que pour le calcul de $expr_2$.

let x =2 in x * x ;;	let a = 1 and b = 2 in 2*a+b ;;
x ;;	let a = 1 and b = 2 * a in b + a;;
let y = 3 in x + y ;;	let a=1 in let b=2*a in b+a ;;
y;;	
let y = (let x = 3 in x + 3) ;;	
let x =3 in y = x + 3 ;;	
let x=(1 , "coucou") and y=("hello", 2.1) in (snd x , fst y) ;;	

Expressions de conditions :
Syntaxe : **if** $expr_{Bool}$ **then** $expr_1$**else** $expr_2$ **;;**
La valeur de cette expression est la valeur de $expr_1$ si l'expression booléenne $expr_{Bool}$ s'évalue à **true** et la valeur de $expr_2$ sinon (à **false**).

if true then 1 else 0 ;;	let x=3 in if x=0 then 0 else 15/x ;;
(if 3=5 then 8 else 10) + 5 ;;	

Les Fonctions

1. Expressions fonctionnelles (fonctions anonymes).

Définition : Une expression fonctionnelle est constituée d'un (ou plusieurs) *paramètre* et d'un *corps*. Elle est introduite par les mots réservés ***function*** (pour les fonctions à un seul argument) ou ***fun*** (pour les fonctions à plusieurs arguments).

a/ Fonction anonyme à un seul argument :

Syntaxe : ***function*** *p -> expr* ;;

Exemple :

la fonction qui élève au carré son argument s'écrit en CAML :

```
# function x -> x*x ;;
- : int -> int = <fun>
```

L'application d'une fonction à un argument s'écrit comme la fonction suivie par l'argument.

```
# (function x -> x * x) 5 ;;
- : int = 25
```

b/ Fonction anonyme à plusieurs arguments :

Syntaxe : ***fun*** $p_1 \ldots p_n$ *-> expr* ;;

Exemple :

```
# fun x y -> 3*x + y ;;
-: int -> int -> int = <fun>
```

L'application d'une fonction à plusieurs arguments s'écrit comme la fonction suivie par les arguments.

```
#(fun x y -> 3*x + y) 4 5 ;;
-: int = 17
```

2. Définitions de fonctions

Comme dans le cas des valeurs simples, il est possible d'assigner un nom à une fonction.

Syntaxe : **let** *nom = expr_fonc* ;;

Où *nom* est le nom de la fonction et *expr_fonc* est une expression fonctionnelle (*function* ou *fun*)

Comme le montrent les exemples suivants (définitions et applications):

```
# let carre = function x -> x*x ;;
carre : int -> int = <fun>
# carre 5 ;;
- : int = 25
```

```
# let f = fun x y -> 3 * x + y ;;
f : int -> int -> int = <fun>
# f 2 4;;
- : int = 10
```

Simplification de la syntaxe:

Pour simplifier l'écriture, la syntaxe suivante est acceptée pour la définition d'une fonction d'arité n (avec n>=1).

Syntaxe : **let** *nom* $p_1 \ldots p_n$ = *expr* ;;

Exemple : les deux fonctions précédentes peuvent être définies comme suit :

```
# let carre x = x*x ;;
carre : int -> int = <fun>
```

```
# let f x y = 3 * x + y ;;
f : int -> int -> int = <fun>
```

3. Ecrire en langage CAML les fonctions suivantes

- Succ qui calcule le successeur d'un entier.
- Pred qui calcule le prédécesseur d'un entier.
- Sum qui calcule la somme de 2 entiers.
- Max qui calcule le maximum de 2 réels.
- Max3 qui calcule le maximum de 3 réels de 2 façons différentes (Sans utiliser Max, puis en utilisant Max).
- MinMax qui donne le min et le max en même temps de 2 entiers.
- Carre qui calcule le carre d'un entier.
- SCarre qui calcule la somme des carrés de 2 entiers (en utilisant la fonction Carre).
- ValAbs qui calcule la valeur absolue d'un entier.
- Abs qui calcule la fonction : Abs (x, y) = | x – y | .
- Surf qui calcule la suface d'un cercle de rayon r ($\prod$ = 3.14).
- Pair qui retourne vrai si son argument est un entier pair, faux sinon.
- Prem qui retourne vrai si son argument est un entier premier, faux sinon.

$x * y = \frac{x}{x} x(2*y)$

Les Fonctions Récursives

1. Définitions de fonctions récursives.

Définition : Une fonction récursive est une fonction dont la définition fait appel à elle-même.

Syntaxe : **let rec** *nom* $p_1 \ldots p_n$ = *expr* ;;

Important : Le mot réservé ***rec*** est obligatoire pour indiquer qu'il s'agit d'une fonction récursive.

Remarque : Pour pouvoir traiter les cas de bases et le cas général, il est utile d'utiliser la structure if-then-else dans la partie *expr* .

L'exemple suivant montre comment définir, en CAML, la fonction récursive « factorielle » nommée *fact :*

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

fact : int -> int = <fun>
```

2. Ecrire en langage CAML les fonctions récursives suivantes :

- Fact : factoriel d'un entier n .
- Exp qui calcule la fonction : Exp (x , y) = x^y .
- Sigma : la somme des entiers de 0 à x .
- Sigma2 : la somme des entiers compris entre 2 entiers a et b.
- Fib : la fonction Fibbonacci définie par :

$$\text{Fib (n)} = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ \text{Fib (n-1)} + \text{Fib (n-2)} & \text{sinon} \end{cases}$$

- pgcd : la fonction qui calcule le PGCD de 2 entiers, en utilisant l'algorithme d'Euclide :

PGCD (a,b) = PGCD(b,r), où r est le reste de la division de a par b.

NB : Vous pourrez utiliser la fonction primitive (a mod b) qui renvoie le reste r.

- pgcd2 : la fonction qui calcule le PGCD de 2 entiers, en utilisant l'algorithme suivant :

$$\text{PGCD (a, b)} = \begin{cases} a & \text{si } a = b \\ \text{PGCD (a-b, b)} & \text{si } a > b \\ \text{PGCD (a, b-a)} & \text{si } b > a \end{cases}$$

- SumSerie :la fonction qui calcule la somme des n premiers termes de la série harmonique de la forme :

1 + 1 /2 + 1/3 + ... + 1/n (pour n = 0, la somme est 0).

TP N°4

EXERCICES GENERAUX

Exercice 1

Ecrire une fonction calculant le produit de deux nombres suivant la méthode de la multiplication dite égyptienne :

$$\text{Mult\_egypt}(x,y) = \begin{cases} 0 & \text{si } x=0 \\ \text{Mult\_egypt}(x/2, 2*y) & \text{si } x \text{ est pair} \\ \text{Mult\_egypt}(x-1, y) + y & \text{sinon} \end{cases}$$

Exercice 2

Ecrire une fonction calculant le ppcm (plus petit commun multiple). On pourra s'aider du pgcd (plus grand commun diviseur).

Exercice 3 :

1. Ecrire une fonction qui inverse un mot. Exemple "CAML" sera transformé en "LMAC"

2. Ecrire une fonction qui reconnaît les palindromes, comme "RADAR" ou "eluparcettecrapule "

Exercice 4 : Bissextile

Ecrire et tester la fonction de détermination des années bissextiles utilisant la définition suivante :

« Toutes les années divisibles par 4 sont bissextiles sauf celles qui sont divisibles par 100 et qui ne sont pas divisibles par 400. »

Exercice 5 :

1- Ecrire une fonction récursive qui calcule le reste de la division entière de x par y.

2- Ecrire une fonction récursive qui calcule le quotient de la division entière de x par y.

Remarque : On ne pourra pas utiliser les opérations / et mod

TP 5 CaML Light

Les Listes

1/ Définition de la liste :

Les listes sont des structures permettant de regrouper un nombre quelconque d'éléments de même type. Exemples :

Liste d'entiers	Liste de réels	Liste de chaines de car	Liste de listes d'entiers	Liste vide
[14 ; 5; 0 ; 658]	[12.5 ; 25.8]	["ali" ; "omar" ; "kader"]	[[1 ; 6] ; [5] ; [] ; [12 ; 25 ; 8]]	[]
Type : int list	Type : float list	Type : string list	Type : int list list	Type : 'a list

2/ Les Opérations sur les listes :

Deux opérations sont prédéfinies en CAML :

La concaténation de liste, notée @	L'ajout d'un élément en tête de liste, notée ::
# [1 ;5] @ [2;8] ;; - : int list = [1; 5; 2; 8]	# 5 :: [3;8] ;; - : int list = [5; 3; 8]
# [] @ [2;8] ;; - : int list = [2; 8]	# (1 ::(2 ::(3 :: [ ]))) ;; - : int list = [1; 2; 3]

Deux fonctions sont prédéfinies en CAML :

fonction ***hd*** : retourne le 1er élément de la liste	fonction ***tl*** : retourne la liste sans le 1er élément
# hd [12; 25 ; 18] ;; - : int = 12	# tl [12 ;25 ;18] ;; - : int list = [25; 18]

3/ Quelques fonctions utilisant les listes :

# let f x y = x @ y ;; f : 'a list -> 'a list -> 'a list = <fun>	# let g x y = x :: y ;; g : 'a -> 'a list -> 'a list = <fun>
# f [12; 2] [20] ;; - : int list = [12; 2; 20]	# g 22 [12; 2 ; 10] ;; - : int list = [22; 12; 2 ; 10]
# f ["toto";"tata"] ["titi" ; "tutu"] ;; - : string list = ["toto"; "tata"; "titi"; "tutu"]	# g "toto" ["tata" ; "titi" ; "tutu"] ;; - : string list = ["toto"; "tata"; "titi"; "tutu"]

Ecrire en langage CAML les fonctions suivantes :

1. Calcule le nombre d'éléments d'une liste (***nbElm***).
2. Calcule la somme des éléments d'une liste (***somElm***).
3. Calcule la moyenne des éléments d'une liste en utilisant les fonctions ***nbElm*** et ***somElm***.
4. Insère un élément au début d'une liste.
5. Insère un élément à la fin d'une liste.
6. Supprime un élément au début d'une liste non vide.
7. Supprime un élément à la fin d'une liste non vide (utiliser la fonction ***nbElm***).
8. Inverse une liste.
9. Projette l'élément n°I (I >0) d'une une liste non vide.
10. Détermine si un élément donné appartient ou non a une liste.
11. prend un nombre entier et une liste d'entier et compte les occurrences de ce nombre dans cette liste.
12. Calcule la fonction ***Map*** définie par : ***Map f*** [a1 ; a2 ; ... ; an] -> [(*f* a1) ; (*f* a2) ; ... ; (*f* an)]
    (Distribue la fonction *f* sur les éléments de la liste).
    - Appliquer la fonction ***Map*** pour élever au carrée les éléments d'une liste d'entiers.
    - Appliquer la fonction ***Map*** pour inverser les mots dans une liste de mots.
13. Trie une liste d'entiers par ordre croissant.