Expression Logique et Fonctionnelle . . . Évidemment

TP n° 1 : Premiers contacts avec OCaml

1 Prise en main de l'environnement de programmation de OCaml

1.1 Les commandes des différents compilateurs de OCaml

`ocaml`	boucle d'interaction
`ocamlrun`	interprète de code-octet
`ocamlc`	compilateur en ligne de code-octet
`ocamlopt`	compilateur en ligne de code natif

Table 1 – Commandes de compilation

1.2 Les conventions de nommage de fichiers

extension	signification
`.ml`	fichier source
`.mli`	fichier interface
`.cmo`	fichier objet (code-octet)
`.cma`	fichier d'une bibliothèque objet (code-octet)
`.cmi`	fichier interface compilé
`.cmx`	fichier objet (natif)
`.cmax`	fichier d'une bibliothèque objet (natif)

Table 2 – Extensions des fichiers

1.3 La boucle d'interaction

1.3.1 Démarrer l'interprète

On lance la boucle d'interaction avec l'interprète de OCaml avec la commande `ocaml`

```
$ ocaml
        Objective Caml version 3.08.2

#
```

Le symbole # est une invite signalant que l'interprète attend une phrase à évaluer.

Pour un confort d'utilisation accru de la boucle d'interaction, en particulier pour bénéficier de l'accès à l'historique des commandes passées à l'interprète lors d'une session avec les touches d'édition, nous vous recommandons d'invoquer l'interprète avec la commande `ledit ocaml`, au lieu de la simple commande `ocaml`.

1.3.2 Quitter l'interprète

On quitte l'interpète avec la directive `#quit` ou en tapant la combinaison de touches `Ctrl+D`.

1.3.3 Charger un fichier source

Pour charger un fichier source dans la boucle d'interaction, on utilise la directive `#use`.

Si le fichier `fact.ml` contient les phrases suivantes

```
let rec fact = function n ->
  if n=0 then 1
  else n*fact(n-1)

let _ = Printf.printf "%d\n" (fact 6) ;;
```

alors l'utilisation de la directive `#use` dans la boucle d'interaction produit les réponses :

```
# #use "fact.ml" ;;
val fact : int -> int = <fun>
720
- : unit = ()
#
```

et toutes les déclarations faites dans le fichier sont définies dans l'environnement courant.

On peut aussi lancer l'exécution de l'interprète sur l'ensemble des phrases contenues dans un fichier :

```
$ ocaml fact.ml
720
$
```

ou en plus bavard

```
$ ocaml < fact.ml
        Objective Caml version 3.08.2

#     val fact : int -> int = <fun>
# 720
- : unit = ()
#
$
```

Dans ce cas, après évaluation de toutes les phrases, on quitte l'interprète.

1.4 Emacs et OCaml

Il existe plusieurs modes Caml pour Emacs. Ces modes offrent
– une coloration syntaxique des sources,
– l'envoi de phrases à évaluer depuis le buffer d'édition vers le buffer de l'interprète.
Le mode conseillé est le mode `Tuareg`. Pour l'installation de ce mode, voir la rubrique documents du site du cours.

Démarrer l'interprète `M-x run-caml` depuis Emacs.
Transmettre une phrase d'un buffer vers l'interprète `C-x C-e`.
Transmettre le contenu d'un buffer vers l'interprète `C-c C-b`.

2 Petits exercices

Exercice 1

Question 1 Testez ces phrases en OCaml.

```
132 ;;
125.78 ;;
1+2 ;;
```

Question 2 Déclarations de valeurs – entiers.

```
let x = 2 ;;

(*inspection de la valeur*)
x;;
let x = 3 ;;
```

```
x ;;
let y = x * x ;;
let x = x * x ;;
(*  autre exemple  *)
let x = 4 * 7 + 3 ;;
let y = (x - 29) * x ;;
let a = x - y
and b = x + y ;;
a;;
```

Question 3 Arithmétique avec les flottants : ne fonctionne pas avec l opérateur + mais avec +.

```
 4.0 + 3.0 ;;
 4.0 +. 3.0 ;;
```

Question 4 Déclarations locales.

```
let pi = 3.14
and r = 2.0  in
    2. *. pi *. r
```

Question 5 Un type construit : les paires.

```
let x = 1 ;;
let y = 2 ;;
(x,y) ;;
(1,2,"elfe") ;;
let xf = 1.0 ;;
let yf = 2.0 ;;
(xf,yf) ;;
```

Question 6 Une liste d entiers.

```
[x ; y ; x] ;;
```

Question 7 L'opérateur = n'est pas une affectation.

```
xf = 4.0 ;;
xf ;;
x ;;
x = 2 ;;
```

Question 8 Portée des variables.

```
let ma = 7 in
  let la =
     let ma = 2 in
          ma + 1 in
    ma + la ;;

(* un autre *)
let x = 1 ;;
x ;;
let z =
  let x = x + x in
    let x = x + x in
       x + x;;
z ;;
x ;;
```

Question 9 Expressions conditionnelles.

```
 if false then 5 else 7
 if true then 5 else 7
 if 4 < 2 then 3*5 else 7 * 1
 if 4 = 2*2 then 3*5 else 7 *1
```

Question 10 Fonctions.

```
(* abs: Z -> N
        a -> -a si a < = 0
             a sinon
*)

(* SYNTAXE: function <param> -> <expr> *)
function a -> if a < 0 then -a else a;;
(function a -> if a < 0 then -a else a) -3 ;;
(function a -> if a < 0 then -a else a) (-3) ;;

(* deux declarations globales equivalentes *)
let abs =  function a -> if a < 0 then -a else a;;

(* sans le mot cle function *)
let  abs a =  if a < 0 then -a else a

(* attention lorsqu'on melange les deux styles...*)
let abs a =  function a -> if a < 0 then -a else a;;


(* fonctions and parentheses *)
let carre x = x * x;;
carre(4);;
carre 4 ;;
carre (2 + 3);;
carre 2 + 3 ;;
(carre 2 ) + 3 ;;
carre 2 + carre 3 ;;
carre (2 + carre 3);;
carre (carre 3);;
```

Question 11 Exemples de fonctions du livre 'apprentissage de la programmation avec Ocaml' de dubois et menissier-morain, page 46.

```
(* calcul polynomial *)
let monpoly x =
  let a = 2
  and  b = -3
  and c = 5
  in
    a*x*x + b*x +c;;

(* des fonctions avec des strings *)
let repeat x = x ^ x ;;
repeat "les vacances";;
let rallonge x = x ^ " c'est bien";;
rallonge "les vacances";;
rallonge repeat "les vacances";;
rallonge (repeat "les vacances");;
let vide x = x="";;
vide "j'ai enfin tout compris !";;

(* declaration de fonction, variable, et portee statique... *)
let p = 10;;
let k x = (x,p,x+p);;
k 1003;;
k p ;;
let p = 1003 ;;
k p ;;
(* question: pourquoi 10 en deuxieme position, et non 1003? *)



(* un exemple de calcul....
```

```
    fait de differentes manieres... procedures auxiliaires *)
  (* calculer puissance 8 pour la valeur 2*)
  let a = 2*2;;
  let b = a*a ;;
  let c = b*b ;;
  (* abstraire le calcul pour la valeur x *)
  let a x = x * x ;;
  let b x = a x * a x ;;
  let puissance8 x = b x * b x ;;
  puissance8 2;;
  puissance8 3;;
  b 2;;
  (* quelle est la portee des noms de fonctions a, b et puissance8 ?
     du point de vue logiciel est-ce une bonne facon de faire ? *)
  (* comment limiter la visibilite des fonctions auxiliaires de puissance8? *)

  (* EXO: re-definir puissance 8 avec fonction auxiliaires locales a etb*)

  (* autre definition. avec fonction auxiliaire qu'on veut bien
garder dans l'environnement global *)

  let carre x = x * x ;;
  let power8 x = carre (carre (carre x));;
  power8 2;;
  power8 3;;

  (* fonction avec deux parametres *)
  let average a  b = (a +. b) /. 2.0 ;;
  average 3.0 4.0 ;;
  average 3.0 ;;
```

Question 12 Une fonctionelle sur les listes.

```
let double x = x * 2
in
  List.map double [31 ; 19;  3];;
```

Question 13 Fonctions mutuellement recursives pair/even et impair/odd.

```
(* ajoutez les parentheses si necessaire *)
let rec even x = x mod 2 = 0
and odd x = not even x ;;
```

Question 14 Puissance 7 avec 4 multiplications sans modifier l'environnement.

```
let a = 13 in
let a2 = a * a in
let a4 = a2 * a2 in
  a * a2 * a4
```

Et sans même utiser de déclaration locale de variables.

```
(function a ->
  (function a2 ->
    (function a4 -> a * a2 * a4
    ) (a2 * a2)
  ) (a * a)
) 13
```

Question 15 Signe d'un entier... corrigez, complétez.

```
let  sign x =
  if x=0
  then  0
  else (if (x>0) then x / x )   ;;
```

Question 16 xor : contrôlez les parenthèses et mots-clés !

```
let xor a b = (not a and b ) or (a and not b);;
```

Question 17

```
let rec fibo n =
  if n = 0
  then 0
  else
     if n = 1
     then 1
     else fibo (n-1) + (fibo (n-2)) ;;
```

Exercice 2

Question 1 Déterminez le type de la fonction `String.length`. Puis utilisez cette fonction en respectant son type. Que calcule cette fonction ?

Question 2 Déterminez le type de la fonction `String.sub`. Puis utilisez cette fonction en respectant son type. Que calcule cette fonction ?

Question 3 En utilisant les deux fonctions `String.length` et `String.sub`, définissez les fonctions

1. `initiale` de type `string` → `string` qui donne l'initiale de la chaîne passée en paramètre ;
2. `saufInitiale` de type `string` → `string` qui donne la chaîne passée en paramètre privée de son initiale ;
3. `finale` de type `string` → `string` qui donne le caractère final de la chaîne passée en paramètre ;
4. `saufFinale` de type `string` → `string` qui donne la chaîne passée en paramètre privée de sa finale ;
5. `miroir` de type `string` → `string` qui donne la chaîne miroir de celle passée en paramètre.

Exercice 3

Réalisez une fonction nommée **puissance** de type `int` → `int` → `int` pour calculer la puissance n-ième d'un entier.

3 Le code de César

Le code de César est un système de chiffrement utilisé par Jules pour communiquer des informations secrètes à ses généraux. Il consiste à décaler toutes les lettres de trois rangs dans l'alphabet. Ainsi un `A` devient un `D`, un `B` devient un `E`, ..., un `Z` devient un `C`, et le message

`LE PONT NEUF FAIT SOIXANTE PIEDS.`

devient

`OH SRQW QHXI IDLW VRLADQWH SLHGV.`

Vous allez réaliser un programme qui effectue ce codage.

3.1 Décalage dans l'alphabet

Les littéraux caractères en OCaml sont dénotés entre 'apostrophes'. Les fonctions `int_of_char` et `char_of_int`, de types respectifs `char` → `int` et `int` → `char`, convertissent un caractère en son code ASCII (nombre entier compris entre 0 et 255) et vice-versa.

Question 1 Réalisez une fonction `decale` de type `int` → `char` → `char` qui décale tout caractère qui est une lettre majuscule ou minuscule[1], et laisse inchangé tout autre caractère, l'importance du décalage étant donné par le premier paramètre. Ci-dessous quelques exemples d'utilisation de cette fonction

```
# decale 3 'e' ;;
- : char = 'h'
# decale 5 'W';;
- : char = 'B'
# decale 7 ':' ;;
- : char = ':'
```

Vous pourrez utiliser avec profit l'opérateur **mod**, et le fait que dans le code ASCII toutes les lettres majuscules sont situées entre le `A` et le `Z`, et toutes les minuscules entre le `a` et le `z`.

1. ne considérez pas les lettres accentuées telles que é, è, ...

3.2 La fonction cesar

Question 2
En utilisant
- la fonction `String.get`, de type `string → int → char`,
- la fonction `String.make`, de type `int → char → string` qui construit une chaîne de caractères d'une longueur donnée remplie avec un caractère donné,
- l'opérateur `^` de concaténation des chaînes de caractères,
- et certaines fonctions que vous avez programmées dans un exercice précédent,

réalisez la fonction `cesar` de type `int → string → string` dont voici un exemple d'utilisation :

```
# cesar 3 "Le pont neuf fait soixante pieds." ;;
- : string = "Oh srqw qhxi idlw vrladqwh slhgv."
```

Question 3 Quelle expression permet de décoder un message codé avec `cesar`[2] ?
Question 4 À quelle focntion correspond la fonction définie ci-dessous ?

```
let que_fais_je = function s ->
  let g = cesar 17 in
    cesar 9 (g s)
```

3.3 Production d'un exécutable

Dans cette partie, on suppose que toutes les déclarations sont faites dans un fichier nommé `cesar.ml`.
Question 5 À la fin de ce fichier ajouter le code la procedure `main` ci-dessous.

```
let main () =
  let n = int_of_string Sys.argv.(1)
  and s = Sys.argv.(2) in
    Printf.printf "%s\n" (cesar n s) ;
    exit 0

let _ = main ()
```

Cette procédure, de type `unit → unit`, récupère les deux premiers arguments passés dans la ligne de commande (shell), et les communique à la fonction `cesar`. Le résultat est affiché sur la sortie standard.
Question 6 Pour compiler le fichier `cesar.ml` et obtenir un fichier exécutable, tapez la commande qui suit dans un terminal[3].

```
$ ocamlc -o cesar cesar.ml
```

Vous obtiendrez ainsi un fichier exécutable nommé `cesar`, et il suffit de l'invoquer dans un terminal sous la forme

```
$ ./cesar 3 "Le pont neuf fait soixante pieds."
Oh srqw qhxi idlw vrladqwh slhgv.
```

pour obtenir le décalage de trois rangs.

2. Prenez garde au fait qu'en Caml, l'entier résultant de l'utilisation de l'opérateur **mod** a le même signe que le dividende.
3. cette commande suppose que le répertoire courant contient le fichier `cesar.ml`