Université de Bordeaux
Année 2017-2018

Mastère 2
T.A.P

TD n°1 - Premiers pas avec OCaml

Exercice 1: Utilisation de l'environnement

1. Lancer la boucle d'interaction OCaml avec la commande `ocaml`.
2. Expérimenter un par un les points suivants :
   - Le symbole `#` est le symbole d'invite de l'environnement. Il signifie que le système est en attente d'une entrée de l'utilisateur.
   - Un "retour chariot" après un double point-virgule `;;` indique que le système doit compiler et évaluer l'expression qui le précède.
   - La définition de nouveaux symboles se fait à l'aide du mot-clé `let` par la construction `let ⟨symb⟩ = ⟨expr⟩`.
   - La définition de nouvelles fonctions se fait aussi à l'aide du mot-clé `let` en utilisant par exemple la construction `let ⟨symb⟩(⟨var1⟩, ... , ⟨varn⟩) = ⟨expr⟩`.

▷ Le développement d'un programme peut se faire entièrement sous l'environnement interactif. Néanmoins, pour plus d'efficacité, il est conseillé d'utiliser `emacs` avec le mode Tuareg (`M-x tuareg-mode`). Dans ce mode, il est possible d'envoyer une expression à la boucle d'interaction avec la commande `C-x C-e`.

Exercice 2: Liens de développement

1. La page du cours : `http://dept-info.labri.u-bordeaux.fr/~narbel/TAP/`
2. La page de développement du langage : `http://caml.inria.fr/`
3. La page de la documentation/manuel :
   `http://caml.inria.fr/pub/docs/manual-ocaml/index.html`
4. "Les bosses" : `http://caml.inria.fr/cgi-bin/hump.en.cgi`
5. "Développement d'applications avec Objective Caml" :
   `http://caml.inria.fr/pub/docs/oreilly-book/`

Exercice 3: Tester la boucle d'interaction

1. Utiliser la boucle d'interaction comme une calculatrice (se référer à la page de documentation `the core library`, le module `Pervasives` pour obtenir les opérateurs disponibles).
2. Vérifier la différence entre les opérateurs sur les entiers et sur les flottants.
3. Définir de nouveaux identificateurs :
   (i) Définir x comme étant la valeur `true` et l'évaluer.
   (ii) Redéfinir x comme étant la fonction `sqrt` et l'évaluer.
   (iii) Définir la fonction $2 * x + 3$.
   (iv) Définir la fonction $3 * \cos^2(x) + 1$.
   (v) Définir la fonction "valeur absolue" à l'aide d'une conditionnelle.

Remarque : La syntaxe des expressions conditionnelles en OCaml est la forme usuelle :

`if ⟨expr⟩ then ⟨ok-case⟩ else ⟨ko-case⟩`

Exercice 4: Fonctions de première classe

1. Écrire une fonction `make_even` qui construit la partie paire d'une fonction f :

$$f_{\mathsf{paire}} : x \mapsto \frac{f(x) + f(-x)}{2}$$

2. Écrire une fonction `deriv` approximant la dérivée d'une fonction f en un point x, à l'aide du paramètre ϵ non nul de la manière suivante :

$$f'_{\epsilon}(x) : x \mapsto \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

   Noter dans chacun des cas le type de la fonction obtenue.

3. Comment est-il possible de composer ces deux fonctions, pour extraire par exemple la partie paire de la dérivée de la partie paire d'une fonction f ?

Exercice 5: Problèmes de récursivité

La construction `let rec ⟨symb⟩(⟨var`$_1$`⟩, ... , ⟨var`$_n$`⟩) = ⟨expr⟩` permet d'indiquer au compilateur que la fonction `⟨symb⟩` est définie de manière récursive.

1. Programmer la fonction `fact` qui calcule le nombre $n!$ de manière récursive.

2. Programmer la fonction `newton` qui applique l'algorithme de Newton à une fonction f. Rappelons que l'algorithme de Newton consiste à rechercher un zéro d'une fonction en l'approximant par la suite :

   $$\begin{cases} u_0 = x \\ u_{n+1} = u_n - f(u_n)/f'(u_n) \end{cases}$$

   Veiller à réfléchir aux arguments nécessaires à l'algorithme.

   Remarque : le point fixe de la fonction cosinus en radians est de l'ordre de 0.739.

Considérons maintenant la fonction suivante :

```
let approx_pi n =
  let rec approx_pi_rec n =
    if (n = 0) then 0 else
      let a = approx_pi_rec (n-1) in
      let u = Random.float 1. and v = Random.float 1. in
        if (u*.u+.v*.v<=1.) then (a+4) else a
  in (approx_pi_rec n,n);;
```

Cette fonction approxime la valeur de π en utilisant un algorithme de Monte-Carlo.

3. Quelle est à votre avis la valeur maximale de n pour laquelle il est possible d'appliquer cette fonction ? Pourquoi cela ?

4. Le code suivant réalise un calcul très similaire au précédent, avec une différence (à peine) subtile. En quoi cette différence pourrait-elle résoudre le problème rencontré à la question précédente ?
   Est-ce qu'elle le résout effectivement ?

```c
#include <stdlib.h>
#include <stdio.h>

int approx_pi_rec(int n, int s) {
  float a,b;

  if (n==0)
    return(s);
  else {
    a = ((float) random())/RAND_MAX;
    b = ((float) random())/RAND_MAX;
    if ((a*a+b*b)<=1)
      return(approx_pi_rec(n-1,s+4));
    else
      return(approx_pi_rec(n-1,s));
  }
}

void approx_pi(int n) {
  int s = approx_pi_rec(n,0);
  printf("Approx : %d/%d\n",s,n);
}

int main(void) {
  approx_pi(10000);
  approx_pi(100000);
  approx_pi(1000000);
  approx_pi(10000000);
  return 1;
}
```

Exercice 6: Problèmes d'abstraction

Considérer la fonction `stable_sort` définie dans le fichier fourni dans les sources. Cette fonction permet d'ordonner une liste d'éléments selon l'ordre défini par les opérateurs `<` et `<=`. Néanmoins, en toute généralité, il est possible de vouloir ordonner une liste selon une fonction d'ordre autre que celle définie en OCaml.

Modifier cette fonction en lui rajoutant un paramètre qui soit une fonction de comparaison (on pourra s'inspirer du prototype de la fonction `compare`). Compléter la modification en faisant que cette fonction permette de choisir l'ordre qui est utilisé pour ranger la liste. Tester votre fonction en ordonnant une liste d'entiers dans l'ordre décroissant.

Remarque : il n'est pas nécessaire de comprendre l'algorithme de tri pour faire l'exercice, simplement d'arriver à distinguer à quel endroit les tests de comparaison sont effectués.