Bonjour

Méthodologie de Programmation avec Objective CAML

Vincent Poirriez

courriel :Vincent.Poirriez@univ-valenciennes.fr

Objectifs

Programmation fonctionnelle

Programmation fonctionnelle et/ou impérative

Récursif ou itératif

Programmes et structures polymorphes

Gestion des cas exceptionnels Comment le faire proprement ?

Programmation par contrat

Structurer le code Pour faciliter la réutilisabilité et la maintenance.

Développement en équipe

Organisation du cours de Méthodologie de Programmation

Intervenants et planning

Cours

Vincent Poirriez	18 h	12 x 1h30 le vendredi de 10h30 à 12h00

TD

Vincent Poirriez et Dorian PETIT	18 h	11 x 1h45 le Mardi de 13h45 à 15h30 (Groupe 1), le Mardi de 15h45h à 17h30 (Groupe 2).

TP

Samuel COLIN et Dorian PETIT Samuel.Colin@univ-valenciennes.fr Dorian.Petit@univ-valenciennes.fr	12 h	5 x 2h30 le mercredi de 13h30 à 16h (Groupe 1), le Mercredi de 16h à 18h30 (Groupe 2).

Ce qui est attendu de vous

Vous réferrez au conseil de certificat pour la répartition entre examen, TP et contrôle continu.

Le contrôle continu (CC) sera constitué de

- Un devoir surveillé (DS) comptant pour 70% du contrôle continu.
- Des devoirs à la maison (DM) qui s'échelonnerons sur toute la durée de l'enseignement. Ces travaux seront à rendre électroniquement. Ils comptent pour 30% du contrôle continu.
- La formule de calcul de CC est :
  CC= max DS (0.7 *. DS +. 0.3 *. DM)

Les travaux pratiques seront notés par les intervenants de TP. La première scéance est une scéance de prise de contact.

Premier Cours

Présentation générale

Le système et ocaml et sa documentation

OCAML : Un langage portable et porté

Disponible sur de nombreux OS
Compilateurs distribués gratuitement par l'inria depuis 1984.

Localement :

`http://www.univ-valenciennes.fr/LAMIH/ROI/poirriez/ensuvhc/ocamluvhc/index.html`.

Ou, le site référence. Pour récupérer sources, manuels, compléments :

- http ://caml.inria.fr/index-fra.html/
- http ://www.ocaml.org/

Les ouvrages vivement recommandés

- Le langage Caml, Pierre Weis et Xavier Leroy,
  DUNOD, Paris 1999 (deuxième édition)
- Développement d'applications avec Objective Caml
  E.Chailloux, P.Manoury, B.Pagano, O'REILLY 2000
- Approche fonctionnelle de la programmation,
  Guy Cousineau et Michel Mauny, Ediscience (Collection Informatique),
  Paris 1995

Il y en a d'autres . . .

Les langages fonctionnels

Historique

Au début des langages de programmation, deux modèles de programmation coexistent :

Le modèle de Von Neumann du quel sont sortis les langages impératifs.

Le Lambda Calcul modèle théorique des langages fonctionnels.

LISP : premier langage fonctionnel proposé dans les années 60.

De nombreux langages ont suivi, au début handicap d'efficacité

Aujourd'hui, grâce aux progrès de la compilation, de nombreux compilateurs efficaces pour des langages fonctionnels existent.

Domaines d’utilisation

Ils sont utilisés dans de nombreuses applications(emacs, gimp, autocad, jeux (langage SCOL), téléphonie (langage ERLANG), démonstrateurs. . .)

Domaines de prédilection :

– manipulation de symboles (compilation, traduction, intelligence artificielle, documents web,. . .) ;
– conception de logiciels complexes (grâce au niveau d’abstraction),
– apprentissage de la programmation.

Dans la famille ML je demande…

Histoire de la famille ML

- L'ancêtre : ML = meta-langage du projet LCF(1980)
- Les premières machines abstraites :
  - La FAM : Cardelli
  - La CAM : Curien, Cousineau
- Une spécification théorique : 1994 (Milner)
- Plusieurs implantations : CAML : (1987) ; SML/NJ : (1988)
- Les différentes branches évoluent constamment.Pour CAML :
  - Caml-light :1990 (Xavier Leroy, Damien Doligez)
  - Modules paramétrés et code natif : CSL (1995)
  - Extension objet : (Didier Rémy et Jérôme Vuillon)(1996)
  - Étiquettes documentant les arguments et arguments optionnels (Jacques Garrigue)(1999)
  - . . .

Choix de OCAML comme langage support

Caractèristiques générales

- Possède une sémantique simple, compréhensible et rigoureuse
- Autorise de nombreux modes de programmation

  fonctionnel basé sur la notion de calcul

  impératif basé sur les notions d'état et d'effet

  orienté objet

  multi-threadée avec des processus légers inclus

  programmation par filtrage

  il est possible d'utiliser conjointement ces différents modes.
- communique sur le réseau Internet,
- indépendant de l'architecture machine.

Facilite une programmation sûre

Écriture et Compilation

- Typage statique fort : vérifie des propriétés ensemblistes.
  - Typé statiquement sans nécessité d'annotations (qui restent possibles).
    Garanti une exécution sans erreur dues au typage
  - Typage polymorphe (par le compilateur).
    Permet un grand taux de réutilisation sans alourdir la tâche du programmeur.
  - Déclaration obligatoires des types complexes.
- Programmation par cas, vérification de l'exhaustivité.
- Gestion claire des erreurs et des cas exceptionnels.
- Des bibliothèques puissantes.évite de réinventer la roue

Facilite une programmation sûre

Exécution

- Gestion automatique de la mémoire
- Déclenchement d'exceptions
  et non génération d'un bus error core dump par :
  - Vérification dynamique de la validité des accès

    Cette vérification dynamique automatique est un atout de sécurité.
    Elle a un coût.
    Elle est débrayable (option de compilation de ocamlopt) au risque et péril de l'utilisateur.
  - Vérification de l'exhaustivité du filtrage

Adapté pour des petits et des grands projets

- petits projets :
  - Interfacé avec l'éditeur `emacs`
  - permet la mise au point rapide
  - Outils de mise au point : `trace, printf, deboggueur avec retour arrière`
- Grands projets
  - Un système de modules très puissant
  - Compilation séparée
  - Ouvert au monde : interface aisée avec le langage C
  - utilisation des Makefile

Deux modes de compilation

Production de bytecode :

- hautement portable et mobile ;
- utilisable dans une boucle interactive ;
- interprèté par une machine virtuelle.

Production de code machine :

- Très efficace (compilateur optimisé) ;
- Dépendant de la configuration : architecture et OS.

La boucle interactive ocaml

Comme de nombreux langages fonctionnels,

Objective Caml dispose d'une boucle interactive.

– Permet de mettre au point des petits bouts de programmes sans se soucier des interactions.
– Très pratique.
– Attention, quand le programme est mis au point, si on veut pouvoir l'exécuter en dehors de la boucle interactive, il faut prévoir les entrées sorties.
– Les phrases entrées dans la boucle interactive *doivent* être terminées par `;;`[a].

[a] C'est optionnel en dehors de la boucle interactive si toute les phrases sont des déclarations.

entrer et sortir de ocaml

```
marsiq> ocaml
                Objective Caml version 3.06

# exit 0 ; ;
marsiq>
```

Les premiers pas avec Caml

Mise en oeuvre

Attention, la façon la plus commode d'écrire des programmes en Objective Caml est :

– utiliser l'éditeur **`emacs`** (ou **`xemacs`**).
– avec un mode **`ocaml`**
– visualiser le source dans un buffer
– la boucle interactive dans un autre buffer.

Ainsi, la mise au point du programme est incrémentale.

Programmer comme l’on pense

la fonction PGCD en C

```c
/* suppose que a >= b */
#include <stdio.h>
int pgcd(int a, int b)
 {
    int r ;
   while (r = a % b) {
      a = b;
      b = r;
   }
   return b;
}
```

Modification d’états,preuves par invariants . . .

La fonction PGCD en Caml

```
(*pgcd.ml*)
let rec pgcd a b =
  let r = a mod b in
  if r = 0 then b
  else pgcd b r
```

Implémentation directe, pas de preuve à faire sur les états.

– `rec` dit : fonction récursive. C'est le seul mot informatique.

– `let x = e in` signifie `soit x égale à e dans ce qui suit`.

– Pas de ~~return~~.

– Dans un premier temps vous pouvez penser `pgcd(a,b)` au lieu de `pgcd a b`

Un exécutable en C

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]){
   int a,b,r,tmp;
   a= atoi(argv[1]);
   b= atoi(argv[2]);
   if (a < b) {
      tmp = a;
      a = b;
      b = tmp;}
   printf ("le pgcd est %d\n",pgcd(a,b));
   exit (0);}
```

puis `gcc -o pgcd1 pgcd.c mainpgcd.c`

Un exécutable en CAML

```
(*mainpgcd.ml*)
open Printf
open Pgcd
let point_d'entrée =
  if !Sys.interactive then () else
  let a = int_of_string Sys.argv.(1)
```

puis `ocamlc -o pgcd2 pgcd.ml mainpgcd.ml`

Points importants

Tout programme est une suite de

– Définitions : types, valeurs,. . .

définition de valeur	`let x = e`
définition de fonction	`let f x0 x1 ... xn = e`
définition de fonctions mutuellement récursives	`let [rec] f1 x1 ... = e1 ... [and fn xn ... = en]`
définition de type(s)	`type q1= t1... [and qn= tn ]`

– La dernière étant le point d'entrée du programme, fonction **`main`** en C, sans nom imposé en Objective Caml.

Ces définitions peuvent être dans différents fichiers et/ou modules.

La factorielle biensûr

afficher la factorielle de 12

```
1  (*fact12.ml*)
2  let rec fact(x) =
3   if x <= 1 then 1 else fact(x-1) * x
4  let affiche =
5   print_int (fact 12);
6   print_newline();;
```

Le point d'entrée.

Remarquer le `;;` qui termine une phrase.

– Obligatoire pour finir les phrases dans la boucle interactive
– Optionnel dans les fichiers, conseillé en fin de fichier.

Compilé puis exécuté par : `crabe>ocamlc -o f12 fact12.ml`

```
crabe> f12
479001600
```

Remarque : Aucune information de typage.

OCAML, un langage de script

Éditer un fichier `script1.ml` par exemple :

```
1  (* script1.ml*)
2  let bienvenue =
3    "Bonne␣idée␣d'utiliser␣Ocaml\n"^
4    "Vous␣travaillez␣dans␣le␣répertoire:␣"^
5    (Sys.getcwd())^
6    "\n␣Votre␣système␣d'exploitation␣est:␣"^
7    (Sys.os_type)^"\n"
8  let point_d'entrée=
9    print_string bienvenue;flush stdout;;
```

puis :

```
$ ocaml < script1.ml # équivalent à la version interactive
```

ou `$ ocaml script1.ml #idem, mais suppression des`

messages

Constructions de base du langage

Types de base, valeurs et fonctions

- nombres
  - int $[-2^{30}, 2^{30} - 1]$ (arithmétique 32)
  - float (IEEE 754) mantisse 53bits, exposant $e[-1022, 1023]$
- caractères : char 'a'
- chaînes de caractères : string "bonjour"
- booléens : bool true et false

Opérations sur les nombres

Entiers		Flottants	
+	addition	+.	addition
-	soustraction	-.	soustraction
*	multiplication	*.	multiplication
/	division entière	/.	division
mod	modulo	**	exponentiation

Ce sont des opérateurs infixes. Ils ont une version préfixe :

```
let additioner (x,y) = ( + ) x y
```

notez

Opérations booléennes

not	négation	Synonymes	
&&	et	&	
||	ou	or	
=	égalité structurelle	<	inférieur
==	égalité physique	>	supérieur
<>	négation de =	<=	inégalité large
!=	négation de ==	>=	inégalité large

Ce sont des opérateurs infixes. Ils ont une version préfixe...

Les opérateurs de comparaisons sont **polymorphes**.

```
val ( = ): 'a -> 'a -> bool
val ( > ): 'a -> 'a -> bool
```

lire quote a flèche quote a flèche bool

compare deux valeurs de même type et renvoie un booléen.

Expressions conditionnelles

if cond then expr1 else expr2

– cond est de type bool
– expr1 et expr2 doivent être de même type.

Déclarations simultanées, ou combinées :

Simultanées :

```
let toi = 1 and moi = 2
let nous = toi + moi
```

nous existons l'un sans l'autre, mais aussi ensembles

Combinées :

```
let eux =
  let lui = 1 and elle = 2
  in
  lui + elle
```

eux sont fusionnels, lui et elle n'existent pas en dehors d'eux

Un dernier premier exemple

Les racines d'une équation du second degré

```
1  (*racs2nd.ml*)
2  let liste_des_racines (a,b,c) =
3    if a = 0. then if b = 0. then [] else [-.c /. b]
4    else
5      let delta = b ** 2. -. 4. *. a *. c in  (* partage*)
6      if delta < 0. then []
7      else if delta = 0. then [-.b /. 2. *. a]
8      else [(-.b +. sqrt(delta)) /. 2. *. a;
9            (-.b -. sqrt(delta)) /. 2. *. a]
10 (*[liste_des_racines (a,b,c)] renvoie la liste
11     des racines du polynôme ax^2 + bx + c*)
```

Le résultat est une liste

Importance du nommage pour partager (voir `delta` ligne 5).

Mérite d’être testée dans la boucle interactive.

Prévoir les interactions

```
12 let _ =
13   if !Sys.interactive then () else
14   let a = float_of_string  Sys.argv.(1)
15   and b = float_of_string  Sys.argv.(2)
16   and c = float_of_string  Sys.argv.(3)  in
17   let rac = liste_des_racines (a,b,c)    in
18   match rac  with                              (* filtrage  *)
19   | [] -> Printf.printf "Pas de racine\n"
20   | [db] -> Printf.printf "Une racine double %f\n" db
21   | [x1;x2] ->Printf.printf "deux racines %f %f\n" x1 x2
22 ;;
```

Remarquez un filtrage : match...with...(ligne 18)