Programmation fonctionnelle

L2 Info et Math-info, 2018–19

Marc Zeitoun

7 septembre 2018

Plan

Objectifs de l'UE

- Comprendre la notion de **récursivité**.
  - Concevoir des programmes récursifs **simples** et **corrects**.
  - **Prouver** certaines propriétés de programmes.
  - Évaluer leur **coût** en temps et en mémoire.

Objectifs de l'UE

- Comprendre la notion de **récursivité**.
  - Concevoir des programmes récursifs **simples** et **corrects**.
  - **Prouver** certaines propriétés de programmes.
  - Évaluer leur **coût** en temps et en mémoire.

- Introduction à la programmation fonctionnelle en **Ocaml**.
  - Très peu de syntaxe.

Points importants pour réussir

- Travail personnel : 2 à 3h/semaine en plus des enseignements.

Points importants pour réussir

- Travail personnel : 2 à 3h/semaine en plus des enseignements.
- Ne pas apprendre par cœur des solutions sans les comprendre.

Points importants pour réussir

- Travail personnel : 2 à 3h/semaine en plus des enseignements.
- Ne pas apprendre par cœur des solutions sans les comprendre.
- Réfléchir **sur papier** à :
  - La **correction** de vos programmes.

Points importants pour réussir

- Travail personnel : 2 à 3h/semaine en plus des enseignements.
- Ne pas apprendre par cœur des solutions sans les comprendre.
- Réfléchir **sur papier** à :
  - La **correction** de vos programmes.
  - L'**efficacité** de vos programmes.

Points importants pour réussir

- Travail personnel : 2 à 3h/semaine en plus des enseignements.
- Ne pas apprendre par cœur des solutions sans les comprendre.
- Réfléchir **sur papier** à :
  - La **correction** de vos programmes.
  - L'**efficacité** de vos programmes.
- **Tester** vos programmes.

Plan

Organisation

- 6 cours (CM) de 1h20

Créneaux variables !.

Organisation

- 6 cours (CM) de 1h20  **Créneaux variables !**.
- 10 cours intégrés (CI) de 1h20.
- 11 TD machine (TM) de 1h20.

Organisation

- 6 cours (CM) de 1h20  **Créneaux variables !**.
- 10 cours intégrés (CI) de 1h20.
- 11 TD machine (TM) de 1h20.

- Groupes de TM : diffusés progressivement, semaine du 10/9, sur `http ://lstinfo.emi.u-bordeaux.fr`.

Équipe pédagogique

uf-info.ue.prog-fonc@diff.u-bordeaux.fr

- **Info A1** : Silvia Pagliarini (en anglais)
- **Info A2** : Marc Zeitoun
- **A1–A2 TM** : Irène Durand

- **Info A3** : Frédérique Carrère
- **Info A4** : Thomas Place
- **A3–A4 TM** : Philippe Duchon

- **Info A5** : Théo Pierron
- **Math-Info A1** : Irène Durand
- **Math-Info A2** : Bernard Serpette
- **MIA1–MIA2–A5 TM** : Thomas Place

Supports d'enseignement

Le cours est fait en CM **et en CI**.

- Les TM sont importants.

Supports d'enseignement

Le cours est fait en CM **et en CI**.

- Les TM sont importants.

- Site **Moodle** de l'UE, ouverture fin de semaine.
  - **Exercices** de TD et sous Moodle.
  - **Polycopié** à lire **pendant** le semestre.
  - **Diaporamas** de cours et **code commenté** tapé en cours.
  - **Sujets des examens** des années précédentes.
  - Plusieurs **supports** (livres pdf) sur `ocaml.org`.

Évaluation

Session 1 : **0.5** Examen + **0.5** Contrôle continu

Session 2 : **0.5** Examen + **0.5** max(Examen, Contrôle continu)

Évaluation

Session 1 : **0.5** Examen + **0.5** Contrôle continu

Session 2 : **0.5** Examen + **0.5** max(Examen, Contrôle continu)

Contrôle continu

	Date	**Durée**	**Coefficient**
Test en CI	Semaine du 24/9/2018	25mn	15%
Test en CI	Semaine du 15/10/2018	25mn	15%
DS	16/11/2018	1h15	40%
TP noté en TM	Semaine du 3/12/2018	1h20	30%
Moodle	Tout le semestre	–	bonus

Comment réussir

- Lire et **comprendre** le cours avant les CI et TM
- Être **actif/active** en CI et TM
- **Chercher** les exercices **soi-même**
- Ne pas hésiter à nous contacter par mail
- Utiliser le **forum** questions/réponses de Moodle.

Contenu de l'UE

- Introduction à **OCaml**.
- **Complexité** en temps et en mémoire des algorithmes.
- Fonctions **récursives**.
- Types **récursifs**.
- Récursivité terminale

Plan de ce premier cours

Voir le polycopié sous Moodle pour plus de détails.

- Pourquoi OCaml ?

Introduction à OCaml.
  - Types
  - Expressions
  - Fonctions et premiers programmes
  - Récursivité.

Plan

Pourquoi OCaml ?

Pour programmer de façon plus sûre

Pourquoi OCaml ?

Pour programmer de façon plus sûre

Logiciels dont les failles peuvent créer des **dommages dramatiques**

- humains,
- pour l'environnement,
- économiques.

Domaines :

- santé
- transports
- énergie
- algorithmes financiers.

Tous les logiciels ne sont pas critiques mais il y a un **besoin** et des **métiers** associés à la production ou la vérification de **logiciels sûrs**.

Bugs logiciels aux conséquences désastreuses (1)

Aéronautique

1962 Perte d'itinéraire de la sonde **Mariner 1** (NASA) au lancement.
2 causes, dont erreur de transcription d'une équation.

Bugs logiciels aux conséquences désastreuses (1)

Aéronautique

1962 Perte d'itinéraire de la sonde **Mariner 1** (NASA) au lancement.
2 causes, dont erreur de transcription d'une équation.

1996 Auto-destruction d'**Ariane 5** (1^{er} vol), 37 sec après décollage.
Cause. Conversion flottant 64 bits vers entier 16 bits.

Bugs logiciels aux conséquences désastreuses (1)

Aéronautique

1962 Perte d'itinéraire de la sonde **Mariner 1** (NASA) au lancement.
2 causes, dont erreur de transcription d'une équation.

1996 Auto-destruction d'**Ariane 5** (1^{er} vol), 37 sec après décollage.
Cause. Conversion flottant 64 bits vers entier 16 bits.

2004 Blocage du robot **Mars Rover Spirit**.
Cause. Trop de fichiers ouverts en mémoire flash.

Bugs logiciels aux conséquences désastreuses (1)

Aéronautique

1962 Perte d'itinéraire de la sonde **Mariner 1** (NASA) au lancement.
2 causes, dont erreur de transcription d'une équation.

1996 Auto-destruction d'**Ariane 5** (1^{er} vol), 37 sec après décollage.
Cause. Conversion flottant 64 bits vers entier 16 bits.

2004 Blocage du robot **Mars Rover Spirit**.
Cause. Trop de fichiers ouverts en mémoire flash.

Médecine

85–87 5 morts par irradiations massives dues à la machine **Therac-25**
Cause. Conflit d'**accès aux ressources** entre 2 logiciels.

Bugs logiciels aux conséquences désastreuses (2)

Télécoms

1990 Crash à grande échelle du réseau **AT&T**, effet domino.
Cause. Toute unité défaillante alertait ses voisines, mais la réception du message d'alerte causait une panne du récepteur !

Bugs logiciels aux conséquences désastreuses (2)

Télécoms

1990 Crash à grande échelle du réseau **AT&T**, effet domino.
Cause. Toute unité défaillante alertait ses voisines, mais la réception du message d'alerte causait une panne du récepteur !

Énergie

2003 Panne d'électricité aux USA & Canada, **General Electric**.
Cause. À nouveau : mauvaise gestion d'**accès concurrents** aux ressources dans un programme de surveillance.

Bugs logiciels aux conséquences désastreuses (2)

Télécoms

1990 Crash à grande échelle du réseau **AT&T**, effet domino.
Cause. Toute unité défaillante alertait ses voisines, mais la réception du message d'alerte causait une panne du récepteur !

Énergie

2003 Panne d'électricité aux USA & Canada, **General Electric**.
Cause. À nouveau : mauvaise gestion d'**accès concurrents** aux ressources dans un programme de surveillance.

Finance

2012 **Bourse** de Tokyo paralysée par un bug.

Bugs logiciels aux conséquences désastreuses (3)

Informatique

1994 Bug du **Pentium FDIV Intel** sur opérations en flottants.
Cause. Algorithme de division erroné (trouvé par Th. Nicely).

Bugs logiciels aux conséquences désastreuses (3)

Informatique

1994 Bug du **Pentium FDIV Intel** sur opérations en flottants.
Cause. Algorithme de division erroné (trouvé par Th. Nicely).

06–08 Clés générées par **OpenSSL** et données cryptées non sûres, impactant les applications l'utilisant (comme ssh).
Cause. Générateur de nombres aléatoires d'OpenSSL cassé.

Bugs logiciels aux conséquences désastreuses (3)

Informatique

1994 Bug du **Pentium FDIV Intel** sur opérations en flottants.
Cause. Algorithme de division erroné (trouvé par Th. Nicely).

06–08 Clés générées par **OpenSSL** et données cryptées non sûres, impactant les applications l'utilisant (comme ssh).
Cause. Générateur de nombres aléatoires d'OpenSSL cassé.

78–95 Faille du protocole d'authentification **Needham-Schroeder**.
Protocole très simple (3 messages échangés).
Cause. Attaque **man in the middle** détectée par G. Lowe.
Utilisé 17 ans avec cette faille.

Bugs logiciels aux conséquences désastreuses (3)

Informatique

1994 Bug du **Pentium FDIV Intel** sur opérations en flottants.
Cause. Algorithme de division erroné (trouvé par Th. Nicely).

06–08 Clés générées par **OpenSSL** et données cryptées non sûres, impactant les applications l'utilisant (comme ssh).
Cause. Générateur de nombres aléatoires d'OpenSSL cassé.

78–95 Faille du protocole d'authentification **Needham-Schroeder**. Protocole très simple (3 messages échangés).
Cause. Attaque **man in the middle** détectée par G. Lowe. Utilisé 17 ans avec cette faille.

09–15 Faille de sécurité dans le système **Linux**. **Cause.** Erreur de programmation (en **C**) d'une fonction d'authentification.

Obstacles à la sûreté : écriture directe en mémoire

Le bug de Grub !

Pendant **6 ans**, les versions de **Linux** ont présenté une très belle faille de sécurité, due à la programmation en **C** de la fonction qui saisit le nom de l'utilisateur.

http ://hmarco.org/bugs/CVE-2015-8370-Grub2-authentication-bypass.html

Importance des logiciels critiques

Des bugs dans des applications courantes peuvent être bénins, mais ont dans certains domaines un **coût élevé** en termes

- humain,
- de sécurité,
- économique,
- d'environnement.

Plan

Problèmes du paradigme impératif

Caractéristiques du paradigme impératif :

Problèmes du paradigme impératif

Caractéristiques du paradigme impératif :

- Affectations
- Pointeurs,
- Conversion de types.

Paradigme impératif et affectation

- Typiquement : langage C.
- **État courant** d'un programme donné par **valeurs de variables**.
- Passage d'un état à un autre par **affectations**.
- Certains langages (C) permettent l'accès direct à la mémoire.

  $\rightsquigarrow$ gestion mémoire délicate.

Le problème de l'affectation

- "Source de nombreuses difficultés et de bugs" (Wikipedia)
- Rend le programme difficile à comprendre.

  Exemples : strcpy.c, fusion.py.

Le problème de l'affectation

- "Source de nombreuses difficultés et de bugs" (Wikipedia)
- Rend le programme difficile à comprendre.

  Exemples : strcpy.c, fusion.py.
- Après avoir programmé en C ou Python, il est surprenant de pouvoir concevoir **facilement** des programmes **efficaces sans affectation**.

Exemple : copie de chaîne de caractères en C

- Les exemples en C ne sont pas à comprendre en détail.
- Ils illustrent que
  - le code est parfois difficile à lire,
  - en C, on peut écrire **n'importe où** dans la mémoire.
  - peu de lignes suffisent à exhiber des bugs délicats.

Exemple : copie de chaîne de caractères en C

- Les exemples en C ne sont pas à comprendre en détail.
- Ils illustrent que
  - le code est parfois difficile à lire,
  - en C, on peut écrire **n'importe où** dans la mémoire.
  - peu de lignes suffisent à exhiber des bugs délicats.

```
char *
strcpy(char *s1, const char *s2)
{
    char *s = s1;
    while ((*s++ = *s2++) != 0)
        ;
    return (s1);
}
```

Obstacles à la sûreté : écriture directe en mémoire

```
/* Pris sur Wikipedia, "Depassement de tampon" */
void foo(char *str)
{
  char buffer[16];
  strcpy(buffer, str);
}

int main(int argc, char *argv[])
{
  if (argc > 1)
     foo(argv[1]);
  return 0;
}
```

./a.out 1234567890ABCDEFG cause une erreur à l'exécution ...
sur certaines machines et pas sur d'autres 😡.

Obstacles à la sûreté : cast implicite

```c
#include <stdio.h>

int main (void) {
  int   x = 3;
  int   y = 5;
  float z = x / y ;

  printf("%d/%d = %f\n", x, y, z);

  return 0;
}
```

- Le **cast** en C correspond à une conversion de type.
- Qu'affiche ce programme pour `z` ?

Obstacles à la sûreté : cast implicite

```c
#include <stdio.h>

int main (void) {
  int   x = 3;
  int   y = 5;
  float z = (float) x / y ;

  printf("%d/%d = %f\n", x, y, z);

  return 0;
}
```

- Et celui-ci ?

Obstacles à la sûreté : cast implicite

```c
#include <stdio.h>

int main (void) {
  int   x = 3;
  int   y = 5;
  float z = 1.0 * x / y ;

  printf("%d/%d = %f\n", x, y, z);

  return 0;
}
```

- Même question.

Obstacles à la sûreté : cast implicite

```c
#include <stdio.h>

int main (void) {
  int   x = 3;
  int   y = 5;
  float z = x / y * 1.0 ;

  printf("%d/%d = %f\n", x, y, z);

  return 0;
}
```

- Une dernière variation.

Obstacles à la sûreté : cast implicite

```c
#include <stdio.h>

int main (void) {
  int   x = 3;
  int   y = 5;
  float z = x / y * 1.0 ;

  printf("%d/%d = %f\n", x, y, z);

  return 0;
}
```

- Une dernière variation.
- **Rappel**. Coût de la mauvaise conversion Ariane : $\sim 500000000\$$
  https ://fr.wikipedia.org/wiki/Vol_501_d%27Ariane_5

Paradigme fonctionnel

- Pas d'état : on peut programmer **sans affectation**.
- Les programmes spécifient directement des « équations ».

Paradigme fonctionnel

- Pas d'état : on peut programmer **sans affectation**.
- Les programmes spécifient directement des « équations ».
- Exemple : longueur d'une liste.

  En Ocaml, la liste [**1**;**2**;**3**] se construit comme `1::[2;3]`.

$$\begin{cases} \text{longueur}(\text{listeVide}) & = & 0 \\ \text{longueur}(a :: reste) & = & 1 + \text{longueur}(\text{reste})) \end{cases}$$

Paradigme fonctionnel

- Pas d'état : on peut programmer **sans affectation**.
- Les programmes spécifient directement des « équations ».
- Exemple : longueur d'une liste.

  En Ocaml, la liste [1 ;2 ;3] se construit comme `1::[2;3]`.

$$\begin{cases} \text{longueur}(\text{listeVide}) & = & 0 \\ \text{longueur}(a :: reste) & = & 1 + \text{longueur}(\text{reste})) \end{cases}$$

- Programme Ocaml : traduction presque mot à mot.

```
let rec longueur l = match l with
    [] -> 0
  | a :: reste -> 1 + longueur reste
```

- Bonus : vérification de types et polymorphisme.

Exemple : la fusion de listes : rappel en Python

```
def fusion(a,b) :
    c = []
    n = len(a)
    m = len(b)
    i = 0
    j = 0
    while i < len(a) and j < len(b) :
        if a[i] < b[j] :
            c.append(a[i]) ; i = i + 1
        else:
            c.append(b[j]) ; j = j + 1
    if i == len(a) :
        for j in range(j,len(b)) :  c.append(b[j])
    else :
        for j in range(i,len(a)) :  c.append(a[i])
    return c
```

Exemple : fusion de listes triées en OCaml

```
let rec fusion l1 l2 =
  match l1, l2 with
  | [],_ -> l2
  | _, [] -> l1
  | a::l'1, b :: l'2 ->
     if a <= b
     then a::fusion l'1 l2
     else b::fusion l1 l'2
```

- Plus facile à comprendre : décrit les **propriétés** de la fusion.
- Plus lisible et plus court.

Affectation : récapitulatif

- Réserver l'affectation aux logiciels dont la spécification contient déjà des aspects dynamiques.
- Pour le reste, privilégier la traduction en termes de définitions (style déclaratif).
- Organiser le code entre modules fonctionnels et parties nécessitant vraiment des effets de bord.

Style fonctionnel

- **Valeurs** = nombres, booléens, structures de données, etc., ET **fonctions**.
- **Effets de bord** (entrées/sorties, affectation) : pas dans ce cours
- On peut **prouver** la correction de certains programmes.

Typage fort (différent de celui de Python)

- Typage des variables
- $+$ cast **explicite**
- Pas de surcharge : `+` pour les entiers, `+.` pour les flottants.
- Les erreurs de typage sont détectées **à la compilation**.
- Beaucoup de bugs sont détectés par le **vérificateur de types**.

Où est utilisé OCaml ?

- **Entreprises** : Facebook, Docker, Bloomberg, Jane Street.
- **Universités**, notamment France et US, mais pas que : http ://ocaml.jp/, twitter : #readcoqart.
- **En France** : CEA, Dassault Systèmes, ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information).
- À **Bordeaux** : Shiro Games.
- Voir `https ://ocaml.org/learn/companies.html`

Plan

Prise de contact avec OCaml

Sous emacs ou utop