Premiers pas en Caml

Objectifs

— Apprentissage des règles d'écriture d'un source Caml,
— Prise en main de l'environnement Objective Caml (saisie de programmes, exécution, tests unitaires),
— Programmation de fonctions récursives.

Réalisations attendues

— A rendre : Exercice 5
— Indispensable : Exercices 1 à 7
— Bonus "ludique" : Exercice 8

1 Environnement de travail

1.1 Composition

L'environnement de travail pour les TP de programmation fonctionnelle est le suivant :
— Visual Studio Code : environnement de développement léger, multi-langage ;
— OCaml Merlin : pour la coloration syntaxique, la complétion, l'aide à la navigation, ...
— ppx_inline_test : pour les tests unitaires ;
— UTop : interpréteur OCaml ;
— Dune : pour compiler, interpréter, exécuter, lancer les tests ...

1.1.1 Visual Studio Code

Site web : `https://code.visualstudio.com/`

Visual Studio Code est un éditeur de code développé par Microsoft pour Windows, Linux et OS X. Il est open source, gratuit et supporte un grand nombre de langages.

1.1.2 OCaml Merlin

Site Web : `https://atom.io/packages/ocaml-merlin`

OCaml Merlin propose des aides à l'écriture de programme OCaml sous la forme de :
— Auto-complétion
— Affichage du type d'une expression sous le curseur
— Trouve toutes les occurrences d'une variable
— Renomme toutes les occurrences d'une variable dans un fichier

— Vérifie la complétude du filtrage
— ...

1.1.3 ppx_inline_test

Site Web : `https://github.com/janestreet/ppx_inline_test`

`ppx_inline_test` permet d'écrire des tests unitaires au fil du code OCaml.

La syntaxe est la suivante :
— `let%test "name" = <boolean expr>` : le test passe si l'expression booléenne s'évalue à `true` et échoue si elle lève une exception ou s'évalue à `false`.
— `let%test_unit "name" = <unit expr>` : le test passe si l'expression s'évalue à `()` et échoue sinon (c'est à dire quand une exception est levée).
— Pour des tests anonymes, on peut écrire `_` au lien d'une chaîne de caractères représentant son nom.

1.1.4 UTop

Site Web : `https://opam.ocaml.org/blog/about-utop/`

OCaml possède un interpréteur interactif de haut niveau. Si vous tapez la commande `ocaml` dans votre shell, vous allez avoir une invite qui vous permet de taper n'importe quelle expression OCaml qui sera compilée et exécutée à la volée.

```
$ ocaml
    OCaml version 4.12.0
# 1 + 1;;
- : int = 2
```

Vous pouvez charger vos librairies, et il est assez facile de jouer avec votre code. Néanmoins vous vous rendrez vite compte que cet interpréteur de base a des limites.

UTop est un autre interpréteur interactif de haut niveau pour OCaml. Il a pour but de rendre l'utilisation plus "user friendly" :
— édition interactive de ligne
— complétion des noms de fonctions et valeurs
— aide à la syntaxe
— ...

1.1.5 Dune

Site Web : `http://jbuilder.readthedocs.io/en/latest/`

Dune (anciennement JBuilder) vise à automatiser les opérations répétitives du développement de logiciel en OCaml telles que la compilation, la génération de documents, à l'instar des logiciels Make ou Ant.

Dune lit les métadonnées liées au projet depuis des fichiers `dune`. Il utilise ces informations pour générer les règles de compilation, installation, exécution,...

Création d'un exécutable Dans un répertoire où il y a le fichier `xxx.ml` contenant le code à exécuter, créer un fichier `dune` suivant :

```
(executable
 ((name xxx)))
```

La commande `dune build xxx.exe` générera l'exécutable à l'emplacement `_build/default/xxx.exe`. Noter que l'extension de l'exécutable est `.exe` quelque soit le système d'exploitation.

Définition d'une librairie Dans un répertoire ajouter le fichier `dune` suivant :

```
(library
 ((name mylib)))
```

Soit les fichiers `mylib.mli` et `mylib.ml` existent et il compilera et construira le module `Mylib`. Si ces fichiers n'existent pas il construira un module `Mylib` composé d'autant de sous-modules que de fichiers `.ml` contenu dans le répertoire courant. Les fonctions seront alors appelables de la façon suivante : `Mylib.Nomficher.nomfonction`.

Lancement des tests Pour lancer les tests définis à l'aide de `ppx_inline_test` il suffit de lancer la commande `dune runtest`.

Dépendances Les exemples précédents sont très simples et supposent que la compilation ne dépend pas d'autres modules. C'est rarement le cas. Prenons le cas du fichier `dune` fourni pour le TP :

```
(library
 (name tp1)
 (inline_tests)
 (libraries graphics)
 (modules fact Affichage tp)
 (preprocess
  (pps ppx_inline_test)))
```

Il construira le module `Tp1`. Comme il n'y a pas de fichier `tp1.mli` ni `tp1.ml`, pour appeler la factorielle on doit faire `Tp1.Fact.fact 5;;`.

La ligne `(inline_tests)` précise qu'il y a des tests unitaires dans le fichier.

La ligne `(libraries graphics)` précise que la compilation nécessite d'importer la librairie graphique d'OCaml.

La ligne `(preprocess (pps (ppx_inline_test)))` précise qu'il faut utiliser le pré-processeur `ppx_inline_test`.

Se référer à la documentation en ligne pour plus de détails.

1.2 Installation

Cet environnement peut être installé facilement sur une machine personnelle.

1. Installer Opam 2.x [1].
   `sh <(curl -sL https://raw.githubusercontent.com/ocaml/opam/master/shell/install.sh)`
2. A l'aide d'Opam installer : OCaml 4.07.1, menhir [2], dune, utop, merlin, ppx_expect [3] et ppx_inline_test :
   `opam switch 4.12.0` ou (si échec) `opam switch create 4.12.0`
   `opam install menhir dune utop merlin ppx_inline_test ppx_expect graphics`
3. Installer Visual Studio Code (cf. site web) ;

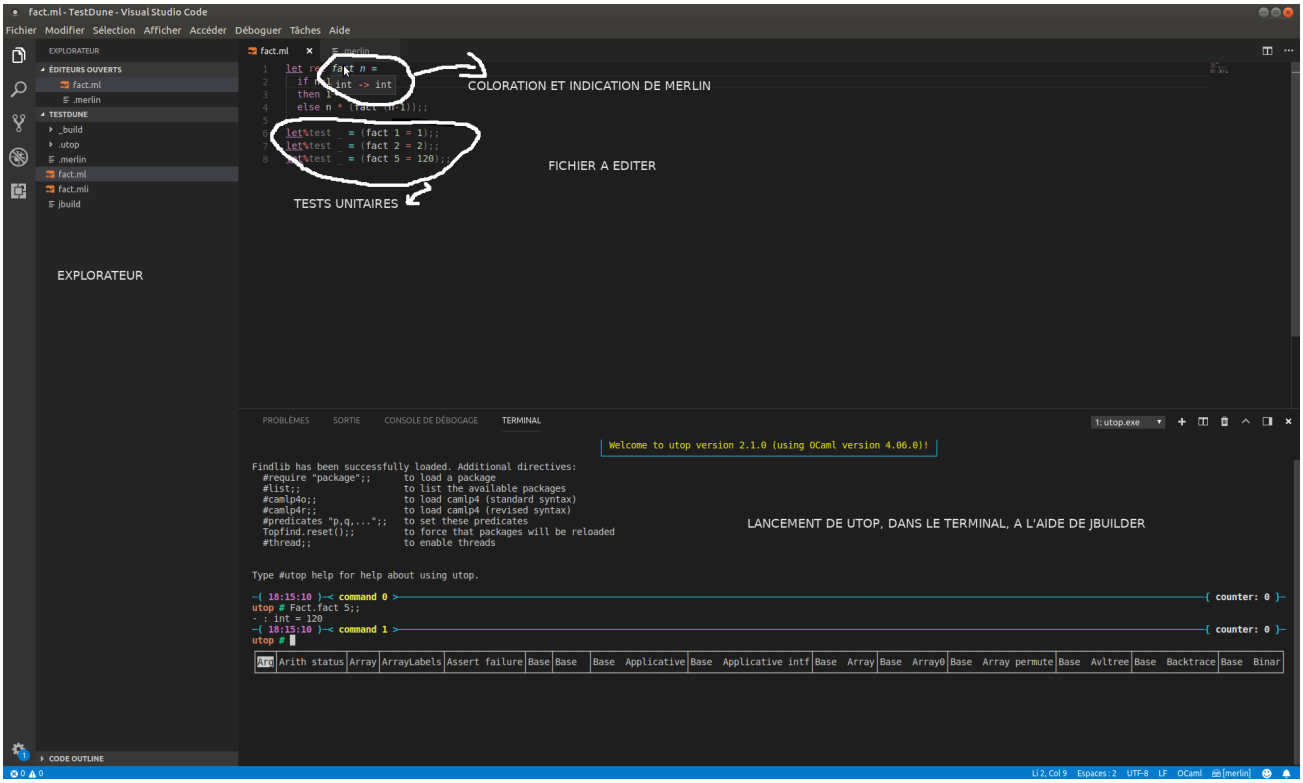
Vous aurez peut être un soucis de dépendance avec m4. Pour l'installer : `apt install m4`.

1.3 Configuration de Visual Studio Code

Commencer par configurer Visual Studio Code dans votre environnement ENSEEIHT en

1. intégrant le plugin OCaml (**barre verticale de gauche** → **extensions** → chercher **OCaml and Reason IDE**) ;
2. intégrant un terminal (**Afficher** → **Terminal intégré**) ;
3. choisir si besoin un thème plus lisible (**Fichier** → **Préférences** → **Thème de couleur**).

1.4 Aperçu



1. `https://opam.ocaml.org/`
2. Utilisé dans seconde partie de l'UE
3. Utilisé dans seconde partie de l'UE

2 Prise en main

▷ **Exercice 1** *Le but de ce premier exercice est de se familiariser avec l'environnement de développement qui sera utilisé tout au long des TPs.*

1. *Mise en place de l'environnement (à ajouter dans le .bashrc pour ne pas avoir à le faire à chaque fois) :*
   ```
   test -r /applications/opam/opam-init/init.sh &&
   . /applications/opam/opam-init/init.sh > /dev/null 2> /dev/null || true
   ```
2. *Récupérer les fichiers du TP1 sur Moodle.*
3. *Dans un terminal, et depuis le répertoire contenant les fichiers du TP1, lancer Visual Studio Code (`code`).*
4. ***Fichier*** → ***Ouvrir le dossier*** → ***Valider*** *(choisir le répertoire avec les sources).*
5. *Ouvrez le fichier `fact.ml`. Les tests unitaires peuvent être surlignés en rouge et sembler non reconnus, cela sera résolu lors de la première compilation.*
6. *Dans le terminal intégré, lancer Utop à l'aide de Dune (`dune utop`). Il ne doit plus y avoir de soucis avec la reconnaissance des tests unitaires.*
7. *Vérifier le calcul de la factorielle de 5 (`Tp1.Fact.fact 5;;`).*
8. *Quitter Utop (`exit 0;;`).*
9. *Lancer les tests unitaires (`dune runtest`) et vérifier qu'aucune erreur n'est levée (cela se traduit par aucun message).*
10. *Ajouter un test erroné et vérifier que les tests unitaires échouent.*

3 Première fonction Caml

Pour les fonctions, on demande de donner les contrats et les tests unitaires.

Le contrat comprend :
- le nom **significatif** et son type
- le rôle de la fonction, expliquée synthétiquement
- le nom **significatif**, le type et le rôle des paramètres, le cas échéant le domaine de validité des paramètres, pour lequel la fonction est bien définie (renvoie un résultat, cas **nominal**), i.e. la **précondition**.
- le type et la spécification du résultat attendu en fonction des paramètres dans le cas nominal, i.e. la **postcondition**.
- la liste des erreurs éventuelles prévues, toujours dans le cas nominal.

Les tests unitaires sont réalisés grâce à l'extension syntaxique `ppx_inline_test`.

La suite du TP est à réaliser dans les fichiers `tp.ml` et `pgcd.ml` pour l'exercice à rendre. Vous pourrez tester à l'aide des tests unitaires fournis ou avec `utop` :
- `dune utop` – lance utop
- `open Tp1.Tp ;;` – charge le module `Tp1.Tp` i.e les fonctions contenues dans le fichier `tp.ml`
- `padovan 2;;` – par exemple pour tester la fonction `padovan` avec la valeur 2

- `open Tp1.Pgcd ;;` – charge le module `Tp1.Pgcd` i.e les fonctions contenues dans le fichier `pgcd.ml`
- `exit 0;;` – pour quitter

▷ **Exercice 2** *Écrire une fonction qui prend en paramètres deux points 2D modélisés par leurs coordonnées flottantes et retourne le coefficient directeur de la droite passant par ces deux points.*

4 Comprendre les types

▷ **Exercice 3** *Ecrire des fonctions de type :*

- `int * int -> bool`
- `int -> bool`
- `'a -> 'a`
- `'a * 'a -> bool`
- `'a * 'b -> 'a`

Vous pourrez tester dans `utop`, que vos fonctions ont bien le bon type, de la façon suivante :

- `dune utop` – lance utop
- `open Tp1.Tp ;;` – charge le module `Tp1.Tp` i.e les fonctions contenues dans le fichier `tp.ml`
- `padovan;;` – ocaml renverra la valeur et type de l'expression `padovan`, si c'est une fonction il affichera `- : int -> int = <fun>`
- `exit 0;;` – pour quitter

▷ **Exercice 4** *Écrire une fonction qui prend en argument un triplet et un index i (entre 1 et 3) et qui retourne la $i_{éme}$ composante du triplet. Quel est son type ?*

5 Premières fonctions récursives

▷ **Exercice 5** ***PGCD***

Un algorithme récursif du calcul du Plus Grand Commun Diviseur repose sur les égalités suivantes :

$$\begin{aligned} pgcd(a,a) &= a \\ pgcd(a,b) &= \begin{cases} pgcd(a-b,b) \ si\ a > b \\ pgcd(a,b-a) \ si\ a < b \end{cases} \end{aligned}$$

*Ecrire et tester la fonction qui calcule le PGCD de deux nombres entiers passés en paramètres. Quelles sont les préconditions ? Peut-on lever ces conditions en utilisant une fonction locale (mécanisme dit d'*encapsulation*) ?*

▷ **Exercice 6** ***Padovan et complexité***

1) *Implanter la fonction qui fournit la valeur du $n^{ième}$ terme de la suite de Padovan ($u_{n+3} = u_{n+1} + u_n$, $u_2 = 1$, $u_1 = u_0 = 0$). Proposer un jeu de test pertinent pour cette fonction.*
2) *Dessiner l'arbre d'appels de la fonction. Noter la redondance des appels. Quel est l'ordre de complexité de cette fonction ?*

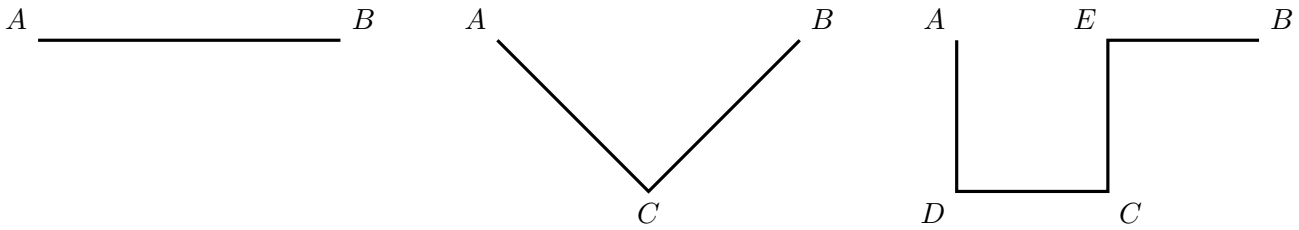
3) On propose de faire une deuxième version de Padovan pour améliorer la complexité :
- *Remarquer que si trois termes consécutifs restent connus, alors la complexité peut être améliorée.*
- *Donner une nouvelle version* `padovan2` *qui a une complexité moindre. Réutiliser les tests de la première fonction* `padovan`*.*
- *Calculer cette nouvelle complexité.*

▷ **Exercice 7** *Ecrire un prédicat qui décide si un nombre entier est premier ou non.*

6 Courbe du dragon

La courbe du dragon est obtenue en suivant le processus ci-dessous :
- le dragon d'ordre 0 sur le segment AB est simplement AB
- le dragon d'ordre n+1 sur le segment AB s'obtient en :
  - plaçant un coin C à gauche de AB, de manière à ce que l'angle ACB soit un angle droit,
  - en faisant un dragon d'ordre n-1 sur les segments AC puis sur BC (attention à l'ordre des points).

Voici les trois premières étapes :

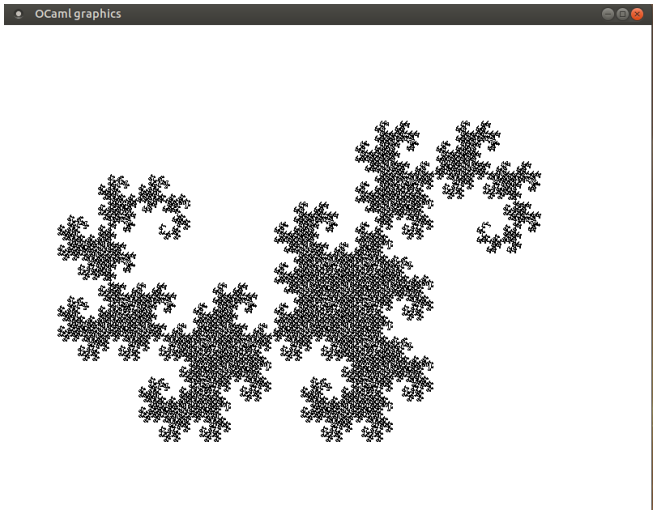


Les coordonnées du nouveau point C sont calculées à partir des coordonnées des anciens points A et B grâce à la formule ((xa+xb)/2 + (yb-ya)/ 2 , (ya+yb)/2 + (xa-xb)/2).

▷ **Exercice 8** *Programmer une fonction* `dragon` *qui dessine la courbe du dragon d'ordre n pour un segment AB.*

Pour pouvoir afficher la courbe du dragon, on a besoin d'utiliser la librairie `Graphics` d'OCaml. La création d'écran est donnée, ainsi qu'une méthode qui permet de dessiner un segment.

Pour un écran de taille 800*600, un point A de coordonnée (200,350), un point B de coordonnée (600,350) et 20 itérations, on doit obtenir le dessin suivant :



Remarque : l'affichage est par définition un effet de bord sur l'écran et fait donc appel à des concepts impératifs. Nous avons choisi de les masquer au maximum pour rester dans un style de programmation fonctionnel.