TRAVAUX PRATIQUES VI

Introduction à OCaml

Pour compiler `file.ml` en un exécutable nommé `exe`, on utilise le compilateur `ocamlopt` :

```
ocamlopt -o exe file.ml
```

On fera **avant tout du OCaml compilé**.

Pour rappel, Ocaml dispose également d'un interpréteur pour tester des morceaux de code. Il s'agit de `ocaml`, mais on lui préfèrera `utop` qui est un bien meilleur interpréteur et possède notamment la possibilité d'éditer vos lignes ou d'accéder à vos précédentes commandes via les flèches du clavier, comme dans votre terminal.

Commençons par vérifier que tout fonctionne avec un fichier simple.

1. Dans un fichier `helloworld.ml`, tapez la commande suivante :

OCaml
```
let _ = print_string "Hello World!\n"
```

Maintenant,exécuter votre code : en compilant [1] : Tapez `ocamlopt -o hello helloworld.ml` puis exécutez le programme avec : `./hello` . .

A Premiers pas

Mentionnons tout d'abord que votre annexe OCaml de cours contient la liste des opérateurs (plus, moins, ou, et, ...) en OCaml. Rappelons également que :

Théorème 1

> Un programme OCaml est une suite de déclarations globales. Le flot du programme est l'évaluation successive des expressions de ces déclarations, dans l'ordre du fichier (de haut en bas).

Remarque : en particulier, pour exécuter une fonction qui travaille par effets secondaires (on parle de procédure), il «faut» [2] utiliser une déclaration :

OCaml
```
let _ = print_string "Hello World!\n"
```

Affichage

Pour afficher des entiers, on utilise l'effet secondaire de la fonction `print_int`. Pour des flottants, `print_float`. Pour des caractères, `print_char`. Et pour des chaines de caractères (du texte), `print_string`.

A.1 Déclaration locale

On peut vouloir faire une déclaration locale de variable (c'est même vital pour survivre). Plus précisément, on peut vouloir déclarer une variable qui soit locale à une expression. Pour cela, on utilise :

`let <identifiant> = <expr0> in <expr1>`

1. Il existe aussi une autre façon de compiler en utilisant `ocamlc helloworld.ml -o hello`. La différence entre `ocamlc` et `ocamlopt` réside dans la nature du fichier produit. `ocamlc` produit un fichier bytecode, intermédiaire entre le code source et les instructions machines, portable mais plus lent. `ocamlopt` crée un fichier natif optimisé pour votre machine, plus rapide à exécuter.
2. Nous verrons plus tard un autre moyen, moins fonctionnel, de le faire.

Ceci **est une expression**. Ce n'est pas une phrase OCaml, c'est une expression. Elle consiste à calculer la valeur de `<expr0>`, l'associe à `<identifiant>`, puis calculer `<expr1>` en substituant dedans les occurences de `<identifiant>` par leur valeur.

Par exemple :

OCaml
```
let x =
  let y = 3 in
  let z = 2*y+1 in
  z -y
```

Ce code est une constituée d'une seule phrase OCaml. Elle donne une valeur à `x` :

— Pour calculer la valeur de l'expression `let y = 3 in let z = 2*y+1 in z -y`, on calcule celle de `let z = 2*3+1 in z -3` (on a remplacé y par 3).

— Pour calculer celle-ci, on calcule d'abord celle de `2*3+1`, puis on substitue dans la suite et on calcule donc la valeur de `7-3` (on a remplacé `z` par `7-3`)

— On conclut alors que la valeur est 4.

A.2 Premières fonctions

Enlevons les roulettes !

On a vu en démo en classe que pour faire une fonction prenant un argument, on utilise `fun x -> ...` . On peut bien entendu prendre d'autres noms de variable que `x`.

2. Dans un même fichier :

a. Déclarer la fonction `cst_0` , fonction constante égale à `0` .

b. Déclarer la fonction `id` , fonction identité [3].

c. Déclarer la fonction `parite` qui prend un entier en argument et s'évalue à `true` si cet est pair et à `false` sinon.

On peut également faire des fonctions prenant deux arguments. Pour cela, on utilise `fun x y ->...` (on peut bien sûr prendre d'autres noms que `x` et `y`). Je vous laisse inférer comment faire des fonctions à trois arguments, à quatre arguments, etc...

3. **a.** Déclarer la fonction `min` qui à deux arguments associe le plus petit des deux selon `<` .

b. Déclarer la fonction `min_3` qui à trois arguments associe leur plus petit selon `<` .

4. Déclarer la fonction `bissextile` qui s'évalue à `true` si l'année passée en argument est bissextile [4] et à `false` sinon.

5. Sans if-then-else, coder une fonction qui teste si un entier est positif.

A.3 Un point important de syntaxe

En Ocaml, si `e` et `f` sont deux expressions, alors `e f` sera **toujours** interpreté comme l'application de `e` à `f`. En particulier, le compilateur va en déduire que `e` est une fonction (et si ce n'est pas le cas, renverra une erreur).

Un exemple un peu caricatural est l'expression `1 2` . Ici, OCaml voudra que `1` soit une fonction (ce qu'il n'est pas, c'est un entier) et l'appliquer à 2.

Si cela vous a l'air d'être trop trivial pour mériter une sous-partie, tant mieux. Mais sachez que vous ferez une erreur de cette sorte à un moment ou un autre durant votre apprentissage.

B Fonctions un peu plus difficiles

À partir de maintenant, l'énoncé pourra préciser le type des fonctions avec la syntaxe `f : type` .

6. Déclarer une fonction `implique` qui prend en entrée deux booléens `a` et `b` et s'évalue en `a` $\implies$ `b` .

7. Que devrait-être la fonction `h` suivante ? Pourquoi cela ne fonctionne-t-il pas ?

OCaml
```
let h = let f = fun x->x in let g = fun x->(-x) in min f g
```

3. C'est à dire $f(x) \mapsto x$.
4. Une année est bissextile si elle est divisble par 4 mais pas par 100. Par contre, les années divisibles par 400 sont quand même bissextiles.

8. Coder la composition de fonction.
9. Implémenter en OCaml les fonctions mathématiques suivantes [5] :
   a. qui à $f : \mathbb{R} \to \mathbb{R}$ associe $\frac{f(0)+f(1)}{2}$
   b. qui à $f : \mathbb{R} \to \mathbb{R}$ associe $f * f$
   c. qui à $f : E \to E$ associe $f \circ f$
   d. qui à $f : \mathbb{R} \to \mathbb{R}$ associe $g : x \mapsto f(x+1)$. Vous proposerez deux versions, avec ou sans composition.
   e. qui à $f : \mathbb{R} \to \mathbb{R}$ associe $\left[f\left(\frac{(f\circ f)(0)+(f\circ f)(1)}{2}\right)+3\right]^4$. Vous utiliserez *uniquement* la composition, l'identité et les fonctions précédentes de cette question comme briques de base. *Il s'agit bien de la puissance 4 et non, pour une fois, d'une note de bas de page.*

C Pour occuper les plus rapides

10. Écrire une expression OCaml qui s'affiche elle-même. On appelle un tel code un programme autoreproducteur. [6]
Pseudo-indication : la fonction printf du module Printf permet d'afficher «comme en C».

5. Adapté d'un TP de Nicolas Pécheux, merci à lui !
6. C'est un concept à la base de beaucoup d'attaques informatiques. La toute première de l'histoire d'internet était un code autoreproducteur laissé libre de se cloner à l'infini sur un réseau ! C'est aussi une histoire sombre, où l'auteur voulait tester la faisabilité de la chose mais n'avait pas conscience des dégats que son virus allait causer… ni que cela finirait au tribunal.