SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

MPIL(LU3IN008)
Année 2019/2020
Séance 1 de TD/TME

Objectifs

1. Premiers pas en programmation fonctionnelle.
2. Les expressions de base.
3. Introduction au typage *fort, statique et par inférence.*
4. Fonctions sur les listes.

Travaux Dirigés

Exercice 1 : Échauffement

Q.1.1 Quelle valeur calcule le programme suivant ?

```
let x = 21;;
let f y = y + 2;;
f x ;;
```

Quel est le type de `x` ? Quel est le type de `f` ? Quel est le type de l'expression `f x` ?

Exercice 2 : Évaluation

Un identificateur (ou nom de variable) est lié à une valeur. Une valeur peut être un nombre entier (`int`), un nombre à virgule flottante (`float`), un booléen (`bool`), une chaîne de caractères (`string`), mais aussi une fonction (`'a -> 'b`), une valeur polymorphe (`'a`), etc.

Par exemple, la déclaration `let x = 42;;` lie l'identifiant `x` à la valeur `42`.

Q.2.1 Donnez le type de chaque identifiant ainsi que sa valeur pour les déclarations suivantes :

```
let a = 42;;
let b = a;;
let a = 23;;
let c = a + b;;
let d = a + int_of_float 2020.01;;
let b = c + b;;
let e = float_of_int d +. 16.64;;
```

Q.2.2 Déclarez un identifiant `f` lié à une fonction prenant un entier et un flottant et retournant la somme des deux nombres. Indiquez également son type.

Q.2.3 (Captures) Indiquez les types des déclarations suivantes, puis donnez la valeur de `b`.

```
let a = 42;;
let f x = a + x;;
let g a = f a;;
let h () = g a;;
let a = 1;;
let i f = g a + f (g a);;
let b = i g + a + g (h()) + f a;;
```

Q.2.4 (Déclarations locales) Indiquez les types des déclarations suivantes, puis donnez la valeur de `b`.

```
let f =
  let a = 42 in
  let f x = a + x in
    f;;
let g a = f 23;;
let h () =
  let a = 9 in
    g a;;
let a = 1;;
let i f =
  let g z = f 2 in
    g a + f (g a);;
let b = i g + a + g (h()) + f a;;
```

Exercice 3 : Les classiques

Vous donnerez les types des fonctions que vous définissez.

Q.3.1 Définissez la fonction identité (celle qui rend son argument intact)

Q.3.2 Définissez une fonction calculant la factorielle

Q.3.3 Définissez une fonction calculant la puissance

Q.3.4 La fonction calculant le `n`-ième nombre de la suite de Fibonacci peut être ainsi définie pour tout nombre entier positif :

$$\begin{cases} \texttt{fib}(0) & = & 1 \\ \texttt{fib}(1) & = & 1 \\ \texttt{fib}(n) & = & \texttt{fib}(n-2) + \texttt{fib}(n-1) \end{cases}$$

Écrivez cette fonction en OCaml.

Q.3.5 Définissez une version de complexité linéaire pour le calcul de n-ième nombre de la suite de Fibonacci.

Q.3.6 Définissez la fonction `map` de type `('a -> 'b) -> 'a list -> 'b list`, telle que `map f l` applique la fonction `f` à tous les éléments de la liste `l`.

Q.3.7 Définissez une fonction `even` rendant true si son argument est pair, false sinon, et une fonction `odd` rendant true si son argument est impair, false sinon.

Si dans votre solution `even` et `odd` ne sont pas mutuellement récursives, redéfinissez-les de manière mutuellement récursive et en n'utilisant pour seule fonction arithmétique que le prédécesseur (`let pred n = n - 1;;`).

Exercice 4 : Types OCaml

Q.4.1 Donnez les types des déclarations suivantes

```
let p = 4. *. atan 1.;;
let circ d = d *. p;;
let a f g = f g;;
let circ_dvd = a circ 12.;;
let print s = print_string (s ^ "\n");;
let check = 2. *. acos 0. = p;;
let id x = x;;
let rec f x = f x;;
let h f x y z = f x (y z);;
let g f x y z = h f x (y z);;
```

Q.4.2 Pour les déclarations suivantes, expliquez les erreurs quand il y en a, et corrigez-les si possible.

```
(*1*) let impr_annee n =
        print_string "Nous␣sommes␣en␣" ^ n;;
(*2*) let imprimer_2020 () =
        impr_annee 2020;;
(*3*) let f x = x x;;
(*4*) let plop = "OCaml␣c'est␣du␣tonnerre";;
(*5*) let plop = plop ^ "␣!" ^ imprimer_2020 ();;
(*6*) let sqrt_delta a b c =
        sqrt (b * b - 4 * a * c);;
(*7*) let reponse_univ = "c'est␣" + 42 + "␣!\n";;
```

Q.4.3 (bonus) Donnez le type de la fonction suivante, puis indiquez ce qu'elle calcule

```
let rec mystere l1 l2 =
 if l1 = [] then l2 else
  mystere (List.tl l1) ((List.hd l1)::l2)
```

Travaux sur Machines Encadrés

Exercice 5 : Prise en main de l'environnement

Q.5.1 (Compilateurs) Il existe deux compilateurs pour OCaml : `ocamlc` et `ocamlopt`.

Le premier, `ocamlc`, produit du code-octet pour une machine virtuelle. Le binaire produit est alors utilisable partout où une machine virtuelle OCaml est disponible.

Le second, `ocamlopt`, produit du code natif optimisé pour l'architecture sur laquelle on se trouve. La phase de compilation est légèrement plus longue mais produit du code très efficace.

Les programmes OCaml sont dans des fichiers avec l'extension `.ml`. Pour les compiler, il suffit d'invoquer l'un des compilateurs avec les fichiers en paramètre.

Par exemple :

```
ocamlc -o monprog f1.ml f2.ml .. fn.ml
```

ou bien

```
ocamlopt -o monprog f1.ml f2.ml .. fn.ml
```

Il est possible de pratiquer la compilation par module pour séparer les étapes de compilation, avec l'option `-c` des compilateurs. Les modules compilés se trouvent dans des fichiers `.cmo` pour `ocamlc` et dans des fichiers `.cmx` pour `ocamlopt`.

Par exemple :

```
ocamlc -c f1.ml
ocamlc -c f2.ml
ocamlc -c f3.ml
ocamlc -o monprog f1.cmo f2.cmo f3.cmo
```

ou bien

```
ocamlopt -c f1.ml
ocamlopt -c f2.ml
ocamlopt -c f3.ml
ocamlopt -o monprog f1.cmx f2.cmx f3.cmx}
```

Q.5.2 (Boucle d'intéraction) Il est possible d'évaluer des expressions dans la boucle d'intéraction (top-level) `ocaml`. Pour cela, exécutez le programme `ocaml` dans un terminal. Vous pouvez y entrer vos déclarations et expressions à évaluer, en les terminant par un double point-virgule. Le top-level vous rendra les résultats au fur et à mesure.

Pour une interface en ligne de commande plus agréable, vous pouvez utiliser l'outil `rlwrap` pour lancer `ocaml` (qui fournit une gestion d'historique et une saisie de texte plus souple).

Exercice 6 : Crible d'Ératosthène

Dans cet exercice, nous allons réaliser un programme qui applique l'algorithme du Crible d'Ératosthène [1] afin de calculer la suite des nombres premiers entre 2 et n.

Q.6.1 (affichage) Écrivez une fonction `print_int_list : int list -> unit` dans un fichier `sieve.ml`, qui affiche les éléments d'une liste d'entiers sur la sortie standard [2] Écrivez un programme qui affiche la liste `[1;2;3;4;5]`, compilez et exécutez-le dans l'environnement de votre choix.

Q.6.2 (intervalle) Écrivez la fonction `interval : int -> int -> int list` telle que `interval n m` calcule la liste des nombres entiers de `n` à `m`. Testez votre solution à l'aide de la fonction `print_int_list`.

Q.6.3 (filtrage) Écrivez la fonction `filter_out : ('a -> bool) -> 'a list -> 'a list` telle que `filter_out p l` retourne la liste des éléments de `l` qui ne satisfont pas le prédicat `p`. Testez votre solution.

Q.6.4 (multiples) Écrivez la fonction `is_multiple : int -> int -> bool` telle que `is_multiple x m` retourne `true` si `m` est un multiple de `x`. Testez votre solution.

Q.6.5 (filtrage des multiples) Écrivez la fonction `remove_multiple_of : int -> int list -> int list` telle que `remove_multiple_of n l` retourne la liste `l` privée des multiples de `n`. Testez votre solution.

Q.6.6 (crible) Écrivez la fonction `sieve : int -> int list` telle que `sieve max` applique l'algorithme du crible d'Ératosthène avec comme valeur maximale `max`. Cet algorithme consiste à créer la liste des entiers de 2 à `max`, puis à supprimer successivement, pour chaque élément de cette liste, les multiples de cet élément. On s'arrêtera dès lors que le carré du plus petit élément de la liste résultante est supérieur à `max`.

Attention : la fonction `sieve` devra probablement faire usage d'une fonction auxiliaire, définie localement, de type `int list -> int list`.

Testez votre solution.

Q.6.7 (Ligne de commande) En déduire le programme tel que l'exécution en ligne de commande de `./sieve <n>` affiche la liste des nombres premiers inférieurs à n.

Remarques : Pour accéder aux arguments du programme, vous pouvez utiliser le tableau `Sys.argv`. Pour accéder à l'élément `n` de `Sys.argv`, il suffit d'écrire `Sys.argv.(n)`.

Exercice 7 : Fonctions de la librairie standard (bonus)

Le module `List` de la librairie standard d'OCaml contient plusieurs fonctions sur les listes fortement utiles. Dans cet exercice, vous redéfinirez les fonctions du module `List` suivantes :

Q.7.1 La fonction `for_all : ('a -> bool) -> 'a list -> bool` qui indique si tous les éléments d'une liste donnée satisfont un certain prédicat. Par exemple, `for_all (fun x -> x > 10) [20;30;40;50] = true`.

Q.7.2 La fonction `map2 : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list` qui applique (deux-à-deux) les éléments contenus dans deux listes distinctes à une fonction d'arité 2. Par exemple, `map2 (fun x y -> x + y) [1;2;3] [4;5;6] = [5;7;9]`. Attention : votre solution devra s'arrêter dès que les éléments de la liste la plus petite auront été tous « consommés » : par exemple, `map2 (fun x y -> x = y) [1;2;3] [0;2;3;9;4] = [false;true;true]`.

Q.7.3 La fonction `combine : 'a list -> 'b list -> ('a*'b) list` qui crée une liste de paires à partir d'une paire de listes. Par exemple, `combine ["rouge";"vert";"bleu"] [1;2;3] = [("rouge",1);("vert'',2);("bleu",3)]`.

Quelques Liens

- Documentation OCaml : `http://caml.inria.fr/pub/docs/manual-ocaml/`
- Index des modules de la bibliothèque standard : `http://caml.inria.fr/pub/docs/manual-ocaml/libref/`
- Développement d'Applications avec OCaml : `https://www-apr.lip6.fr/~chaillou/Public/DA-OCAML/`

1. `https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes`
2. Il sera fait usage de la fonction de la librairie standard `print_int : int -> unit`.