

TP n°1

Premiers pas en OCaml

Les exercices marqués d'une étoile [★] peuvent être laissés pour la fin.

Exercice 1 Pour lancer l'interpréteur OCaml sous emacs :

- ouvrir un fichier `tp1.ml`
- dans le menu Caml, lancer “Start subshell...”
- confirmer le lancement de ocaml

Chaque expression entrée dans la fenêtre de `tp1.ml` peut être évaluée en se plaçant sur un caractère quelconque de l'expression, puis par `:` ou bien “Eval phrase” dans le menu Caml d'emacs, ou bien `ctrl + alt + shift + x`, ou bien `ctrl-c`, `ctrl-e`.

Expressions et types

Exercice 2 Prévoir le résultat fourni par l'interpréteur Caml après chacune des commandes suivantes, entrées dans l'ordre.

```
#let x = 2;;  
#let x = 3  
  in let y = x + 1  
      in x + y;;  
#let x = 3 and y = x + 1  
  in x + y;;
```

Pourquoi les deux dernières commandes ne fournissent-elles pas le même résultat ? Expliquer à présent le comportement suivant :

```
#let x = 3;;  
x : int = 3  
#let f y = y + x;;  
f : int -> int = <fun>  
#f 2;;  
- : int = 5  
#let x = 0;;  
x : int = 0  
#f 2;;  
- : int = 5
```

Exercice 3 Donner le type des expressions suivantes. Vérifier le résultat sur machine.

- `let f x = x + 1 in f 6;;`
- `let f x = x + 1 in f;;`
- `let s = "a" in let f t = t^s in f;;`

Booléens et conditionnelle

Les expressions de type `bool` sont de la forme :

- constantes booléennes :

- `true`

- `false`

- comparaisons :

- `e1 cmp e2`

où l'opérateur est `=`, `<>`, `<`, `<=`, `>`, ou `>=`, avec `e1` et `e2` de même type.

On ne peut pas comparer des fonctions, mais on peut comparer des entiers, des chars, des listes, des n-uplets, etc.

- connections logiques :

- `e1 && e2`, `e1 || e2`, `not e`,

(et, ou, négation de) avec `e1`, `e2`, `e` de type `bool`.

En Caml, tout type non fonctionnel est implicitement muni d'un ordre, et la valeur d'une comparaison entre éléments de même type ne dépend que de cet ordre. Pour les types numériques, il s'agit de l'ordre usuel sur les nombres, pour les `char` l'ordre alphabétique, pour les `string` l'ordre lexicographique, etc.

L'expression suivante a pour valeur la valeur de `e1` si l'expression booléenne `c` vaut `true`, et celle de `e2` si `c` vaut `false` :

`if c then e1 else e2`

Exercice 4 Ecrire une fonction `est_majeur` prenant en argument une année de naissance (un entier), et renvoyant `true` si la différence entre l'année courante et cette année de naissance est au moins égale à 18, `false` sinon.

Exercice 5 Par convention, une année est bissextile : si elle est divisible par 4 et non divisible par 100, ou, si elle divisible par 400. En vous aidant de l'opérateur `mod` sur les entiers, écrivez la fonction `est_bissextile` qui prend comme argument une année (un entier), renvoyant `true` si elle est bissextile et `false` sinon.

Exercice 6 [★] La fonction `max : 'a -> 'a -> 'a` est l'une des fonctions prédéfinies. Appliquée à `x` et `y` de même type, elle renvoie le maximum de `x` et `y` (suivant l'ordre implicite sur les valeurs de ce type, voir ci-dessus).

Ecrire votre propre fonction `max`, en ne vous servant que de `if then else` et des opérateurs de comparaisons. Votre fonction doit être même type que le `max` prédéfini, c'est-à-dire de type `'a -> 'a -> 'a`.

A partir de cette fonction, définir :

- une fonction `max_couple` de type `'a * 'a -> 'a`, prenant en argument un couple `(x, y)`, et renvoyant le maximum de ses composantes,

- une fonction `max_triplet` de type `'a * 'a * 'a -> 'a` prenant en argument un triplet `(x, y, z)`, et renvoyant le maximum de ses composantes.

Récurrence

La construction

```
let rec nom arg_1 = expression ; ;
```

permet de se servir du nom d'une fonction dans sa propre définition, c'est-à-dire de définir des fonctions récursives. Par exemple :

```
let rec fact n = if n <= 0 then 1 else n*(fact (n - 1)); ;
```

Noter que `and` permet de définir simultanément plusieurs fonctions mutuellement récursives :

```
let rec f x = if x > 0 then (g (x - 1)) else 1
      and g x = if x > 0 then (f (x - 1)) else 0
;;
```

Exercice 7 Rappelons que la somme des entiers de 0 à n peut être définie récursivement par $\Sigma(0) = 0$, et $\Sigma(n) = n + \Sigma(n - 1)$ si $n > 0$. Ecrire cette fonction en **CamL**. Que donne cette fonction appliquée à un nombre négatif?

Exercice 8 Ecrire en **CamL** la fonction récursive définie par $f(0) = 1$, $f(1) = 1$ et $f(n) = f(n - 1) + f(n - 2)$.

Exercice 9 [★] Les fonctions en **CamL** sont des valeurs comme les autres. En particulier, une fonction peut prendre en argument une ou plusieurs fonctions, et renvoyer une fonction :

```
let composer f g = (fun x -> f (g x)); ;
```

Ecrire une fonction prenant en argument une fonction f et un entier n , et renvoyant $\sum_{i=0}^n f(i)$. Quel sera son type ? Utiliser cette fonction pour définir une nouvelle fonction calculant la somme des $n + 1$ premiers carrés.

Exercice 10 [★] Ecrire une fonction bi nome $p\ n$ renvoyant C_n^p , donné par :

$$C_n^p = C_{n-1}^p + C_{n-1}^{p-1} \text{ si } 1 \leq p \leq n$$

$$C_n^0 = 1$$

$$C_n^p = 0 \text{ si } p > n$$