Programmation fonctionnelle – TP1

19 février 2010

Utilisation de Ocaml sous Unix. Choisissez votre éditeur préféré pour créer un fichier avec une extension .ml dans lequel vous écrirez votre programme (par exemple tp1.ml). Attention à ne pas utiliser de majuscule pour la première lettre du nom du fichier. Placez-vous dans le répertoire où figure votre fichier puis lancez Ocaml grâce à la commande `ocaml`. Chargez votre programme grâce à l'instruction #`use"tp1.ml";;`. À chaque modification du programme, n'oubliez pas d'enregistrer et de recharger le fichier dans OCaml.

Ex 1. Pour chacun des types suivants donner un exemple de fonction :

1. `'a -> 'a -> 'a -> 'a`
2. `'a -> ('a -> 'a) -> 'a`
3. `('a -> 'b) list -> 'a -> 'b list`
4. `(int -> bool) list -> int -> bool list`
5. `(int*bool) list -> bool`
6. `('a*'a -> 'b) -> 'a -> 'b`

Ex 2. Définir une fonction `abss` qui calcul la valeur absolue d'un nombre entier, par exemple `abss (-2)` évalue à `2`. Définir une fonction `maxx` qui rend le plus grand nombre parmi deux nombres donnés comme argument, par exemple `maxx 3 4` évalue à `4`.

Ex 3. Définir une fonction `pgcd` calculante le plus grand commun diviseur de deux nombres entiers. (*Utiliser la fonction* `abss` *définie précédemment pour gérer les nombres négatifs ; il serait bien apprécié la mis en ouvre de l'algorithme d'Euclide*).

Ex 4. On considère un nombre rationnel comme une pair `(num, den)` de deux nombres entiers, représentante une fraction. Ecrire une fonction `norm` qui normalise une fraction, c'est-à-dire `norm` réduit une fraction telle que numérateur et dénominateur n'ont pas de facteurs en commun. Par exemple `norm (4, 6)` évalue à `(2,3)`. (*Utiliser la fonction* `pgcd` *de l'exercice précédent).*

Ex 5. Définir deux fonctions `somme` et `produit` qui calculent, respectivement, la somme et le produit de deux fractions. On demande que le résultât soit une fraction normalisée.

Ex 6. Ecrire une programme `carre` qui prend en entré une liste des entiers et donne en sortie la liste des carrés des entiers.

Ex 7. Généraliser l'exercice précédent, en donnant une fonction `app` qui s'applique à une fonction arbitraire, donnée comme argument, à chaque élément d'une liste. Par exemple :

`app (fun x -> x*x) [1;2;3]` évalue à `[1;4;9]`

(*Bien sur on ne peut pas utiliser la fonction* `map` *de la librairie standard de OCAML).*

Ex 8. *Ecrire une programme* `somme` *qui prend en entré une liste des entiers et donne en sortie la somme des entiers.*

Ex 9. *En utilisant la fonction* `maxx` *de l'exercice* 2*, écrire une programme* `maxlist` *qui prend en entré une liste des entiers et donne en sortie la maximum entre 0 et les entiers dans la liste.*

Ex 10. *Généraliser les deux exercices précédents, en donnant une fonction* `itera` *qui s'applique à une fonction binaire* `f`*, une liste* `[e1;e2;...]` *et une argument* `x` *d'égal type que les éléments de la liste, et donne comme résultat l'itérations de* `f` *aux éléments de la liste plus* `x`*. C'est-à-dire :*

```
itera f [e1;e2; ... en] x  évalue à  f e1 (f e2 ... (f en x)...)
```

Par exemple :

```
itera add [4;2;5] 0     évalue à     11
```

ou bien

```
itera max [4;2;5] 0     évalue à     5
```

(Bien sur on ne peut pas utiliser la fonction `fold_right` *de la librairie standard de OCAML).*

Ex 11. *Ecrire une fonction* `ordered` *qui prend une liste des nombres entiers en entré et rend en sortie* `true` *si les entiers sont en ordre croissant sinon* `false`*.*

Ex 12 (Difficile). *La suite des nombres de Catalan est définie comme suive :*

$$\begin{array}{ll} C_0 & = 1 \\ C_{n+1} & = \sum_{p+q=n} C_p \times C_q \end{array}$$

Voici les premiers nombres de Catalan $C_0 = 1$*,* $C_1 = 1$*,* $C_2 = 2$*,* $C_3 = 5$*,* $C_4 = 14$*,* $C_5 = 42$*,* $C_6 = 132$*, etc. Les nombres de Catalan sont des entiers naturels qui se rencontrent souvent dans les problèmes de combinatoire, par exemple* C_n *est le nombre des arbres binaires avec* n *noeuds.*

Définir une fonction `catalan` *qui prend un entier* `n` *et donne en sortie l'n-éme nombre de Catalan.*

Ex 13 (Difficile). *Un nombre parfait est un nombre naturel* n *non nul qui est égal à la somme de ses diviseurs stricts, autrement dit, tel que* $n = x_1 + \ldots x_k$ *où* $x_1, \ldots, x_k$ *sont les diviseurs entiers positifs de* n*,* n *non compris. Le premier nombre parfait est* 6*, car* 1*,* 2*, et* 3 *sont les diviseurs stricts de* 6 *et* $1 + 2 + 3 = 6$*. Le deuxième nombre parfait est* 28*, car* $1 + 2 + 4 + 7 + 14 = 28$*. Quel est le troisième nombre parfait ?*

Ecrire une fonction `next-perf` *qui prend en entré un entier* `n` *et donne en sortie le premier nombre parfait plus grande où égal à* `n`*. Evaluer* `next-perf 29` *pour connaître le troisième nombre parfait.*