# TP2 : Impérativité et récursivité

Christophe Rose

Vendredi 1er février 2008

Dans ce TP, nous allons continuer à manipuler les traits impératifs du langage Caml, les boucles et les tableaux. Nous aborderons également la récursivité.

## 1 Boucles et tableaux

**Question 1.** Programmez une fonction `multi` de type `unit -> unit`, qui choisit deux entiers entre 5 et 40, et qui demande à l'utilisateur d'en donner le produit.

Si l'utilisateur répond correctement, une autre question est posée. Sinon, le programme donne la bonne réponse, et indique combien de fois l'utilisateur a bien répondu depuis le lancement du programme.

**Question 2.** À l'aide d'une boucle `for`, programmez une fonction `factorielle` qui prend un entier $n$ et renvoie $n!$.

Un tableau de type `'a` est la donnée $n$ de la longueur du tableau ainsi que les $n$ valeurs de ses éléments.

La commande `make_vect n a` crée un tableau contenant $n$ fois la valeur `a`. La commande `vect_length v` permet de récupérer la taille d'un tableau `v`.

La commande `v.(2) <- b` met la valeur `b` dans la case numéro 2 du tableau, c'est-à-dire la troisième en partant de 0.

**Question 3.** Programmez une fonction `map` de type `('a -> int) -> 'a vect -> int vect` qui prend en argument une fonction $f$ et un tableau $[\![x_1, ..., x_n]\!]$, et qui renvoie $[\![f(x_1), ..., f(x_n)]\!]$.

**Question 4.** Programmez une fonction `somme` et une fonction `produit` qui prennent en argument un tableau d'entiers et qui calculent respectivement la somme et le produit de leurs coordonnées.

**Question 5.** Programmez une fonction qui prend en argument un entier $n$ et qui renvoie le tableau $[\![1, ..., n]\!]$. En déduire une autre façon d'écrire la fonction `factorielle`.

**Question 6.** Programmez une fonction `premier` de type `int -> bool` qui détermine si un entier $n$ est premier ou non. (On pourra faire en sorte de n'utiliser qu'au plus $\sqrt{n}$ fois l'opérateur `mod`.)

**Question 7.** Programmez une fonction `pascal` de type `int -> int vect vect` qui construit un triangle de Pascal. On obtient alors quelque chose de la forme :

```
#pascal 3;;
- : int vect vect =
 [|[|1|]; [|1; 1|]; [|1; 2; 1|]; [|1; 3; 3; 1|]|]
```

## 2 Suite de Fibonacci

La suite de Fibonacci est la suite $(F_n)_{n\in\mathbb{N}}$ définie de manière récursive par $F_0 = 0$, $F_1 = 1$ et $F_n = F_{n-1} + F_{n-2}$ pour $n \geq 2$. Les premières valeurs sont les suivantes : $F_0 = 0$, $F_1 = 1$, $F_2 = 1$, $F_3 = 2$, $F_4 = 3$, $F_5 = 5$, $F_6 = 8$, $F_7 = 13$.

**Question 8.** Écrivez une fonction `fibo_vect` de type `int -> int vect` qui prend en argument un entier $n$ et qui renvoie le tableau $[\![F_0, ..., F_n]\!]$.

Que se passe-t-il lorsque $n \geq 45$ ?

**Question 9.** En utilisant les définitions récursives `let rec f=...`, écrivez une fonction `fibo` de type `int -> int` qui à un entier $n$ renvoie $F_n$.

Que se passe-t-il pour les valeurs de $n$ comprises entre 35 et 44 ? Corrigez le problème rencontré en utilisant une fonction auxiliaire.

**Question 10.** Il est connu que $\lim_{n\to+\infty} \frac{F_{n+1}}{F_n} = \frac{1+\sqrt{5}}{2}$. Utilisez les questions précédentes pour calculer $(1 + \sqrt{5})/2$ à une précision inférieure à `1e-05`.

## 3 Récursivité

**Question 11.** Écrivez une fonction récursive `power` de type `float -> int -> float` qui prend en argument un flottant $x$ et un entier positif $n$ et qui renvoie $x^n$. Le nombre total de multiplications effectuées ne devra pas dépasser $\log_2 n$ environ.

Si vous avez le temps, programmez la fonction de manière impérative, sans récursivité, en utilisant seulement les tableaux et des boucles.

**Question 12.** Écrivez une fonction récursive `pgcd` de type `int * int -> int` qui calcule le PGCD de deux entiers.

**Question 13.** Écrivez une fonction `decompose` de type `int -> unit` qui décompose un entier en produit de nombres premiers. Par exemple,

```
#decompose 36;;
2 2 3 3
- : unit = ()
```

On utilisera les fonctions `print_int`, `print_string` et `print_newline`.

Les listes sont des objets comparables aux tableaux, qui sont définies de manière récursive : une liste est soit la liste vide `[]`, soit un couple `l = t::q` où `t` est une valeur d'un certain type `'a` et `q` une liste de type `'a list`.

**Question 14.** Écrivez une fonction `make_list` de type `int -> 'a -> 'a list` telle que `make_list n x` donne une liste contenant $n$ fois la valeur $x$.

**Question 15.** Écrivez une fonction `longueur_liste` qui détermine la taille d'une liste.

**Question 16.** Est-ce que toutes les listes ont une taille bien définie ?

# 4 Algorithmes de tri

**Question 17.** Écrivez une fonction `random_tab` de type `int -> int vect` qui crée un tableau de taile donnée rempli d'entiers aléatoires. On utilisera la commande `random__int 1000000000;;`.

**Question 18.** Écrivez une fonction `est_trie` de type `'a vect -> bool` qui détermine si un tableau est trié dans l'ordre croissant.

**Question 19.** Écrivez une fonction `algo_ok` de type `('a vect -> 'a vect) -> int -> bool` qui vérifie si une fonction sensée trier un tableau fonctionne sur un tableau aléatoire de taille donnée.

Il existe trois types de tris « en $O(n^2)$ ». Ils s'agit du tri insertion, du tri sélection et du tri à bulles.

Pour effectuer le tri insertion, on considère successivement tous les éléments d'un tableau. On les insère un par un dans un autre tableau, de sorte que ce dernier tableau soit toujours trié. À la fin, le deuxième tableau est trié et contient les mêmes éléments que le premier.

Pour effectuer le tri sélection, on cherche le plus petit élément du tableau, qu'on déplace en première position. Puis on cherche le deuxième plus petit, qu'on déplace en deuxième position. À la fin, tous les éléments sont à leur place.

Pour effectuer le tri à bulles, on considère tous les couples d'éléments successifs, en partant du début du tableau. Si un couple n'est pas trié, on les échange. Lorsqu'on a parcouru tout le tableau, on sait que le plus grand élément est à la fin. À la fin, tous les éléments ont remonté à leur position.

**Question 20.** Écrivez une fonction `echange` telle que `echange v a b` échange les valeurs `v.(a)` et `v.(b)`.

Pour faire le tri insertion, nous allons utiliser le même tableau pour le départ et l'arrivée.

**Question 21.** Programmez une fonction `trouve_pos` telle que `trouve_pos v x k` trouve l'entier $j$ inférieur à $k$ tel que `x` soit compris entre `v.(j)` et `v.(j+1)`, en supposant que le tableau `v` soit trié entre les indices 0 et $k$. En déduire une fonction `tri_insert` de type `'a vect -> 'a vect` qui effectue le tri insertion.

**Question 22.** Programmez une fonction `plus_petit` telle que `plus_petit v k` qui trouve l'indice du plus petit élément de `v` d'indice supérieur ou égal à $k$. En déduire une fonction `tri_select` de type `'a vect -> 'a vect` qui effectue le tri sélection.

**Question 23.** Programmez une fonction `tri_bulle` de type `'a vect -> 'a vect` qui effectue le tri à bulles.

**Question 24.** Vérifiez que les fonctions `tri_insert`, `tri_select` et `tri_bulle` fonctionnent correctement sur certains tableaux, de tailles variant entre 0 et 4000000.

# 5 Structures algébriques

On peut créer de nouveaux types avec Caml, en plus des types prédéfinis. Ainsi, la commande `type complexe = {Re: float ; Im: float}` permet de créer le type « nombre complexe ».

On peut définir des objets de ce type par `let I = {Re: 0. ; Im: 1.}`. On récupère les valeurs d'un nombre complexe `z` en utilisant `z.Re` et `z.Im`.

**Question 25.** Programmez l'addition, la multiplication, la soustraction, le conjugué, le module, l'inverse et la division pour le type « nombre complexe » ainsi créé.

**Question 26.** Créez un type pour $\mathbb{Z}[\sqrt{5}]$, et programmez la multiplication, ainsi que la puissance. Avec la relation $\sqrt{5} \cdot F_n = \left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n$, déduisez-en une fonction qui calcule rapidement la suite de Fibonacci.