TP4 MPSI : Recursivité

fabien.viger@ens.fr
9 mars 2005

But du TP

Dans ce TP, on va enfin exploiter Caml dans sa specialité : la récursivité. Les programmes seront beaux, forts, tout petits.. Pas de questions **bonus** cette fois-ci, mais essayez de faire le TP dans l'ordre .. Et bien sûr, **à n'importe quel moment, toute question est la bienvenue.**

Question 1 : Pourquoi y'a un bug ?

Je vous demande pas de corriger le bug, mais de voir où il est.

```
let rec fibo = function
| x -> fibo (x-1) + fibo (x-2)
| y when y<2 -> 1;;
```

Et dans celui-la ?

```
let rec fibo x = if n=0 then 1 else fibo (x-1) + fibo (x-2);;
```

Question 2 : Listes

En s'aidant eventuellement de l'exemple du programme `print_list` ci-dessous pour se rappeler la syntaxe à utiliser avec les listes, écrire une fonction `len` qui calcule le nombre d'éléments d'une liste.

```
let rec print_list = function
| [] -> ()
| t::q -> print_int t; print_newline(); print_list q;;
```

Question 3 : Max

Écrire la fonction `list_max`, qui trouve le maximum d'une liste. (de type quelquonque). S'aider de la fonction `max` préprogrammée dans Caml. Le code devrait être tout petit..

Question 4 : Map

écrire une fonction `map` de type `('a -> 'b) -> 'a list -> 'b list` telle que `map f [x0; x1; .. ]` renvoie $[f(x_0), f(x_1), \cdots]$.

Question 5 : Max sur un vecteur

Il faut apprendre à "penser récursif" : programmer une fonction *récursive* (sans boucle `for` ou `while`) qui trouve le maximum d'un vecteur. Le mieux est de se servir d'une fonction auxilliaire *aux*, récursive, qui calcule le maximum entre les indices i et j.

Question 6 : Permutation aléatoire

En s'aidant de :

```
let swap_vect v i j = let tmp=v.(i) in v.(i)<-v.(j); v.(j)<-tmp;;
```

Refaire la fonction qui permute un vecteur aléatoirement. Il suffit par exemple de permuter aléatoirement les éléments $[v_0...v_{n-2}]$ puis d'échanger v_{n-1} avec un élément quelquonque du sous-vecteur mélangé. Quelle est la complexité ?

Question 7 : Tri d'une liste

Implémenter le **tri par insertion** sur des listes de type quelquonque. L'algorithme est le suivant : étant donné une liste déjà classée, et un nouvel élément à ajouter à la liste, on va insérer cet élément à sa place.

Question 8 : Definition double

Implementer les suites définies par : $u_{n+1} = u_n/v_n$ et $v_{n+1} = v_n + u_n$, avec $u_0 = 1$ et $v_0 = 1$. Le but est de le faire élégamment, en quelques lignes. Se rappeler qu'avec Caml, il suffit souvent d'écrire les choses de manière presque mathématique! Calculer u_{20}.

Question 9 : PGCD rapide

On peut écrire un PGCD très rapide, uniquement à l'aide de soustractions et de divisions par deux ou de restes modulo 2 (très rapides). L'algorithme se base sur la parité des nombres : par exemple, si a est pair et b impair, $pgcd(a, b) = pgcd(a/2, b)$. En se basant sur ce type de constatations, et en traitant tous les cas possibles, on peut à chaque fois réduire le problème.
Écrire la fonction qui calcule le pgcd rapide

Question 10 : Arbres binaires

On va définir les arbres binaires de recherche. Commencons par le type :

```
type arbre =
| vide
| nd of int*arbre*arbre;;
```

Un arbre est donc soit vide, soit c'est un noeud (`nd`) d'un triplet (x,g,d) ou x est un entier, g un arbre (le sous-arbre gauche) et d un arbre (le sous-arbre droit).
Un arbre binaire de recherche est tel que tous les elements du sous-arbre gauche soit strictement inferieurs a la racine, et tous ceux du sous-arbre droit sont superieurs a la racine.

- Ecrire la fonction permettant de rechercher un élément dans un arbre (de type `int -> arbre -> bool`)
- Ecrire la fonction d'insertion d'un element.
- Ecrire une fonction qui affiche tous les éléments d'un arbre binaire de recherche dans l'ordre.
- Enfin, ecrire une fonction qui affiche une liste d'entiers par ordre croissant, grâce à un arbre binaire de recherche. Quelle est sa complexité ?

Question 11 : Fractales

On va faire des jolis dessins.. Pour ouvrir la fenêtre graphique, taper :

```
#open "graphics";;
open_graph "800x600+0+0";;
```

La fenetre graphique s'ouvre (elle est peut-etre planquée..). On peut à présent faire des dessins.. Essayer :

```
for i=0 to 255 do for j=0 to 255 do
  set_color (rgb i 0 j); plot i j; done; done;;
```

Pour effacer la fenêtre, utiliser `clear_graph()`.

On va céer un crayon virtuel, qui peut avancer en tracant en trait, se repositionner, ou tourner.
Définir la fonction `pos` qui positionne le crayon :

```
let pos x y = moveto x y;;
```

Définir la direction (variable globale)

```
let dir = ref (0,2);;
```

`!dir` est un couple (dx, dy) tel que, quand on fait avancer le crayon, son abscisse augmente de dx et son ordonnée de dy.
Définir la fonction `avance` :

```
let avance () = let (x,y) = current_point() and (dx,dy) = !dir in
  lineto (x+dx) (y+dy);;
```

Et enfin, les fonctions `gauche` et `droite` permettant de tourner :

```
let gauche () = let (dx,dy)= !dir in dir := (-dy,dx);;
let droite () = let (dx,dy)= !dir in dir := (dy,-dx);;
```

À présent, écrire une fonction récursive `dragon` qui fait la chose suivante : $dragon(n, b)$ appelle $dragon(n-1, true)$, tourne à gauche si $b = true$ et à droite sinon, puis appelle $dragon(n-1, false)$.

En même temps, $dragon(0, b)$ ne fait qu'avancer.
Jouer un peu :

```
clear_graph();
moveto 300 300;
dir := (0,10);
dragon 10 true;;
```

puis :

```
clear_graph();
moveto 300 300;
dir := (0,1);
dragon 20 true;;
```

et admirer ...