TD de programmation fonctionnelle et logique
Corrigé du TD 2 : récursivité

10 novembre 1999

Suite et fin de la factorielle récursive et impérative

```
#let rec fact_imp n =
   let a =ref(0)
   in match n with
        0 -> a := 1
      | _ -> let b = ref(fact_imp (n-1))
             in a := n;
   print_int !a;;
             val fact_imp : int -> unit = <fun>
```

Expressions à évaluer

```
#let x = 1;;

#let y = 4;;

#let x = 3 and y = x + 1 in
   let x = x + y and y = x - y
     in (x,y);;

#let x = 3
   in let  y = x + 1
     in let x = x + y
       in let y = x - y
         in (x,y);;
```

```
#let x = 1;;
 val x : int = 1

#let y = 4;;
 val y : int = 4

#let x = 3 and y = x + 1 in
   let x = x + y and y = x - y
     in (x,y);;
     - : int * int = 5, 1

#let x = 3
   in let  y = x + 1
     in let x = x + y
       in let y = x - y
         in (x,y);;
         - : int * int = 7, 3
```

Fonctions récursives simples sur les tableaux

1. Écrivez une fonction qui teste si deux tableaux sont égaux.

```
#let égalité u v =
   let lu = Array.length u and lv = Array.length v
   in let rec comparaison i =
      (i = lu) or ((u.(i) = v.(i)) & (comparaison (i+1)))
      in (lu = lv) & (comparaison 0);;
         val égalité : 'a array -> 'a array -> bool = <fun>
```

2. Écrivez une fonction qui calcule l'élément maximum d'un tableau d'entiers.

```
# let maximum tab =
    let length = Array.length tab
    in let rec maximum_aux i =
       if i = length-1
          then tab.(i)
          else max tab.(i) (maximum_aux (i+1))
       in match length with
            0 -> failwith "tableau vide !"
          | 1 -> tab.(0)
          | _ -> maximum_aux 0;;
                   val maximum : 'a array -> 'a = <fun>
```

Palindromes

Un palindrome est un mot qui se lit bien à l'endroit qu'à l'envers (cf. figure 1).

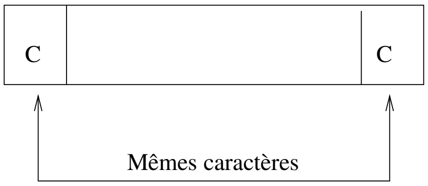

FIG. 1: Structure d'un palindrome

Rappels : l'expression `String.length s` renvoie la longueur de la chaîne de caractères `s` ; l'expression `String.sub s a b` construit une chaîne de `b` caractères égale à la sous-chaîne de `s` débutant par son `a`^{e}caractère.

- Écrire une fonction à valeurs booléennes qui est vrai si le mot qui lui est passé en argument est un palindrome.
  - La chaîne vide est un palindrome.
  - Un mot réduit à un seul caractère est un palindrome.
  - Un mot d'au moins deux caractères est un palindrome si :
    - ses premier et dernier caractères sont identiques ;
    - le mot situé entre ces caractères est un palindrome.

```
# let rec palindrome s =
    match String.length s with
      0 -> true
    | 1 -> true
    | n -> if s.[0]=s.[n-1]
           then palindrome (String.sub s 1 (n-2))
           else false;;
              val palindrome : string -> bool = <fun>
```

```
# let rec palindrome s =
    match String.length s with
      0 -> true
    | 1 -> true
    | n -> (s.[0]=s.[n-1]) & palindrome(String.sub s 1 (n-2));;
          val palindrome : string -> bool = <fun>
```

- Même question mais en ne construisant pas de sous-chaînes de caractères. On utilise deux indices, `i` et `j`, désignant respectivement le début et la fin de la sous-chaîne traitée. Initialement, `i` vaut 0 et `j` la longueur de la chaîne moins 1. À chaque étape on compare les caractères d'indice `i` et `j` :

– s'ils sont égaux on continue ;
– s'ils sont différents on arrête ;
– si i≥j, on a atteint ou dépassé le milieu et on arrête.

```
# let rec palin s i j =
    (i >= j) or ((s.[i]=s.[j])&(palin s (i+1) (j-1)));;
   val palin : string -> int -> int -> bool = <fun>

# let palindrome s = palin s 0 (String.length s - 1);;
 val palindrome : string -> bool = <fun>

# let palindrome s =
    let rec palin i j =
      (i >= j) or ((s.[i]=s.[j])&(palin (i+1) (j-1)))
    in palin 0 (String.length s - 1);;
       val palindrome : string -> bool = <fun>
```

Calcul de (cos(nx), sin(nx))

Écrire une fonction qui prend en entrée un entier n et une paire de valeurs réelles qui sont en fait les valeurs du cosinus et du sinus d'un certain angle x, et qui renvoie la paire (cos(nx), sin(nx)). Autrement dit, le deuxième argument de la fonction est une paire (a,b) telle que a = cos x et b = sin x . Le schéma de calcul doit bien évidemment être récursif.

On pourra se servir des formules de trigonométrie suivantes :

```
cos(nx) = cos((n-1)x) cos(x) - sin((n-1)x) sin(x)
sin(nx) = sin((n-1)x) cos(x) - cos((n-1)x) sin(x)
```

```
# let trigo (a, b) n =
   let rec trigo_aux = function
       0 -> (1.,0.)
     | 1 -> (a, b)
     | m -> let (c,d) = trigo_aux (m-1)
              in (a *. c -. b *. d, a *. d +. b *. c)
   in trigo_aux n;;
           val trigo : float * float -> int -> float * float = <fun>
```