Programmation fonctionnelle et logique
Corrigé du TD1 : récursivité et polymorphisme

Suite de Fibonacci

La suite de Fibonacci est définie comme suit :

$$\mathrm{Fib}(n) = \begin{cases} 1 \text{ si } n = 0 \\ 1 \text{ si } n = 1 \\ \mathrm{Fib}(n-1) + \mathrm{Fib}(n-2) \text{ sinon.} \end{cases}$$

1. Écrire une fonction récursive qui calcule $\mathrm{Fib}(n)$. Écrire deux versions de cette fonction : une qui utilise des tests (expressions booléennes) et l'autre le filtrage.

```
#let rec fib_filtre =
 function
    0 -> 1
  | 1 -> 1
  | n -> (fib_filtre (n-1)) + (fib_filtre (n-2));;
val fib_filtre : int -> int = <fun>
```

```
#let rec fib_filtre n =
 if n <= 1 then 1
          else (fib_filtre (n-1)) + (fib_filtre (n-2));;
val fib_filtre : int -> int = <fun>
```

2. Écrire une fonction récursive qui calcule $(\mathrm{Fib}(n), \mathrm{Fib}(n-1))$. L'utiliser pour écrire une fonction calculant $\mathrm{Fib}(n)$.

```
#let rec fib_paire =
 function
    1 -> (1,1)
  | n -> let (x,y)= fib_paire (n-1) in(x+y,x);;
val fib_paire : int -> int * int = <fun>
```

3. Quelle est la complexité des différentes fonctions ?
Complexité de la première version :

$$C(n) = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{n+1} \right].$$

Complexité de la deuxième version :

$$C(n) = n.$$

Somme des éléments d'un tableau

1. Écrire une fonction qui prend en entrée un tableau et renvoie son élément maximal.

```
#let max t =
  let sup a b = if (a >= b) then a else b in
    let rec max_aux t i m =
      if i = -1
        then m
        else max_aux t (i-1) (sup t.(i) m) in
      max_aux t ((Array.length t)-1) t.((Array.length t)-1);;
val max : 'a array -> 'a = <fun>
```

Tri sur les entiers

1. Écrire une fonction qui prend en entrée un tableau t dont les $n-1$ premiers éléments sont triés et renvoie ce même tableau mais avec les n premiers éléments triés.

```
#let rec insère_aux t n =
  if n = 0
       then t
       else if t.(n) >= t.(n-1)
               then t
               else let temp = t.(n) in
                    t.(n)   <- t.(n-1);
                    t.(n-1) <- temp;
                    insère_aux t (n-1);;
val insère : 'a array -> int -> 'a array = <fun>
```

2. Au moyen de la fonction précédente, écrire une fonction de tri par insertion d'un tableau d'entiers.

```
#let tri t =
  let rec tri_aux t =
     function   0 -> t
              | n -> insère_aux (tri_aux t (n-1)) n in
     tri_aux t ((Array.length t)-1);;
val tri : 'a array -> 'a array = <fun>
```

Tri polymorphe

Écrire une fonction de tri polymorphe sur tableau.

```
#let tri ordre t =
  let rec insère_aux t n =
    if n = 0
         then t
         else if (ordre t.(n) t.(n-1))
                 then t
                 else let temp = t.(n) in
                      t.(n)   <- t.(n-1);
                      t.(n-1) <- temp;
                      insère_aux t (n-1); in
   let rec tri_aux t =
       function   0 -> t
                | n -> insère_aux (tri_aux t (n-1)) n in
     tri_aux t ((Array.length t)-1);;
val tri : ('a -> 'a -> bool) -> 'a array -> 'a array = <fun>
```