Programmation IV

correction de l'examen final

21 juin 2006

Exercice 1 : Mots les plus longs

Partie 1

```
def max(xs: List[Int]): Int =
  (xs foldLeft 0)((a,b) => if (a > b) a else b)
```

Partie 2 (10 pts)

```
def longestWord(ws: List[String]): String =
  (ws foldLeft "")((a,b) =>
    if (a.length > b.length) a else b)
```

Exercice 2 : Sauce au curry

```
def multiplie(a:Int)(b:Int): Int = a * b
val double: X = multiplie(2)
val fois: Y = multiplie
```

Partie 1 : Typage

Les valeurs `double` et `fois` sont des fonction.

```
type X = Int => Int
type Y = Int => Int => Int
```

La syntaxe longue est aussi possible.

```
type X = Function2[Int, Int]
type Y = Function2[Int, Function2[Int, Int]]
```

Partie 2 : Currification explicite

```
def multiplie(x: Int, y: Int): Int = x * y
def curry2[T,U,V](f: (T,U) => V): T => U => V =
  x => (y => f(x,y))
```

```
def double = curry2(multiplie)(2)
double(8)
```

Partie 3 : Retour aux sources

```
def uncurry2[T,U,V](f: T => U => V): (T,U) => V =
  (x, y) => f(x)(y)
uncurry2(curry2(multiplie))(2,8)
```

Partie 4 : Currification en LISP

On applique manuellement en LISP la même transformation que `curry2` applique automatiquement en Scala.

```
(def (multiplie x) (lambda (y) (* x y))
  (val double (multiplie 2)
    (double 8)))
```

Exercice 3 : Iterateur Fibonacci

```
class FibIterator extends Iterator[Int] {
  val hasNext = true
  private var upcomming = 1
  private var current = 0
  def next: Int =  {
    val next = upcomming + current
    current = upcomming
    upcomming = next
    current
  }
}
```

Exercice 4 : Automate à états finis

Partie 1 et 2

```
def run(
  inp: List[Char],
  delta: (Int,Char) => List[Int]
):List[Int] = {

  def next(qs: List[Int], c: Char) =
    qs flatMap { q => delta(q,c) }

  def run1(qs: List[Int], cs:List[Char]):List[Int] =
    cs match {
      case Nil   => qs
      case c::cs => run1(next(qs,c),cs)
```

```
    }
  run1(List(0), inp)
}
```

Partie 3

A tout invocation de `run1`, `qs` est l'ensemble actuel d'états actifs de l'AEF et `cs` la liste des caractères qui doivent encore être lus.

Exercice 5 : Permutations

Partie 1 : En Scala ...

```
def insertion[A](x:A, xs:List[A]): List[List[A]] =
  xs match {
    case Nil     => List(List(x))
    case y :: ys =>
      (x::y::ys)::(insertion(x,ys) map (zs => y::zs))
  }

def permutation[A](as:List[A]): List[List[A]] =
  as match {
    case Nil     => List(Nil)
    case b :: bs =>
      (permutation(bs)) flatMap (xs => insertion(b,xs))
  }
```

Partie 2 : ... et en Prolog

```
insertion(X,[],[X]).
insertion(X,[Y|YS],[X,Y|YS]).
insertion(X,[Y|YS],[Y|ZS]) :- insertion(X,YS,ZS).

permutation([],[]).
permutation([X|XS],ZS) :-
  permutation(XS,YS), insertion(X,YS,ZS).
```