MIAS 1 et MASS 1 – Programmation fonctionnelle

TP n° 6 : corrigé

```
; Exercice 1 : version iterative de puissance : x^n= x^(n-1) * x
; on utilise un compteur variant de 1 a n et on memorise la valeur de x^(k-1)

(define puissance
  (lambda (x n)
    (define aux
      (lambda (k xk)
        (if (= k n)
            xk
            (aux (+ k 1) (* xk x)))))
    (if (= 0 n)
        1
        (aux 1 x))))


; Exercice 2
;          n
; s(n) = somme a^i * b^(n-i)
;         i=0
; version recursive direct : somme(n)= a^n + b*S(n-1)

(define somme1
  (lambda (a b n)
    (if (= n 0)
        1
        (+ (expt a n) (* b (somme1 a b (- n 1)))))))

; version iterative :
; S(k) = a^k + b*S(k-1)
; on utilise un compteur k variant de 1 a n et on memorise les valeurs de a^k et de S(k-1)

(define somme2
  (lambda (a b n)
    (define aux
      (lambda (k ak sk-1)
        (if (> k n)
            sk-1
            (aux (+ k 1) (* ak a) (+ ak (* b sk-1)) ))))
    (aux 1 a 1)))

; Exercice 3
;          n
; S(n) = somme 1/i!
;         i=1
;
; S(k) = S(k-1) + 1/k!
; on utilise un compteur k variant de 1 a n et on memorise les valeurs de S(k-1) et k!

(define somfact
  (lambda (n)
    (define aux
      (lambda (k sk-1 factk)
        (if (> k n)
            sk-1
            (aux (+ k 1) (+ sk-1 (/ 1 factk)) (* factk (+ k 1)) ))))
    (if (= n 0)
        1
        (aux 1 1 1))))
```

```
; Exercice 4 : version iterative de renverse
; idee : on "vide" la liste l1 qui fait office de compteur
; tout en "remplissant" la liste r, parametre accumulateur
; qui est initialement la liste vide

(define renverse
  (lambda (l)
    (define aux
      (lambda (l1 r)
        (if (null? l1)
            r
            (aux (cdr l1) (cons (car l1) r)))))
    (aux l '())))


; Exercice 5 : version iterative de nbocc
; idee : on parcours la liste l1 qui fait office de parametre compteur
; et on compte au fur et a mesure le nombre d'occurence qu'on stocke dans
; le parametre accumulateur r

(define nbocc
  (lambda (o l)
    (define aux
      (lambda (l1 r)
        (cond ((null? l1) r)
              ((equal? (car l1) o) (aux (cdr l1) (+ r 1)))
              (else (aux (cdr l1) r)))))
    (aux l 0)))
```

```
; Exercice 6 : version iterative de concat
; idee : on effectue la concatenation en utilisant la version
; iterative du renversement d'une liste
; on renverse d'abord la liste l2, on obtient une liste rl2
; puis on renverse l1 en prenant comme valeur initiale du parametre
; accumulateur la liste rl2
; on obtient alors une liste rl1l2 qui est la concatenation de l1 et l2 mais renversee
; il reste alors a renverser cette liste pour avoir le resultat desire
;
; Par rapport a la version recursive non terminale utilisant la fonction
; ajoutefin et dont la complexite en temps est quadratique, cette version
; a une complexite en temps lineaire (proportionnelle a la somme des
; longeurs de l1 et l2.

(define concat
  (lambda (l1 l2)
    (define renv
      (lambda (l r)
        (if (null? l)
            r
            (renv (cdr l) (cons (car l) r)))))

    (let* ((rl2 (renv l2 '()))
           (rl1l2 (renv l1 rl2)))
      (renv rl1l2 '()))))
```

```
; Exercice 7 : Calcul iteratif de pi/8
; on utilise un compteur k variant de 0 a n
; on memorise dans le parametre accumulateur pk la valeur de la somme courante
; a chaque etape on augmente pk de u(4k+1)

(define pi
  (lambda (n)
    (define u
      (lambda (i)
        (/ 1 (* i (+ i 2)))))

    (define aux
      (lambda (k pk)
        (if (= k n)
            pk
            (aux (+ k 1) (+ pk (u (+ (* 4 k) 1)))))))
    (aux 0 0)))
```