```
;;;;
;;;; Corrigé.stk        -- Corrigés de la feuille 2
;;;;
;;;;           Author: Erick Gallesio [eg@unice.fr]
;;;;    Creation date: 28-Feb-2000 09:06 (eg)
;;;; Last file update: 28-May-2003 10:55 (eg)
;;;;

;==============================================================================
;                                     EXO 2
;==============================================================================

(define rotation-gauche
  (lambda (l)
    (append (cdr l)
            (list (car l)))))

(define rotation-droite
  (lambda (l)
    (cons (car (reverse l))
          (reverse (cdr (reverse l))))))

;; Cette solution est très inefficace car elle nécessite par trois
;; fois l'appel de la fonction reverse. Une meilleure solution
;; passe par une variable temporaire.
(define rotation-droite
  (lambda (l)
    (define rev (reverse l))
    (cons (car rev)
          (reverse (cdr rev)))))

;; Noter que l'on ne peut pas ôter le dernier reverse car il travaille sur une
;; liste différente.

;; Pour terminer une solution plus efficace, mais moins évidente

(define rotation-droite
  (lambda (l)
    (define aux
      (lambda (l res)
        (cond
         ((null? l)       res)
         ((null? (cdr l)) (cons (car l) (reverse res)))
         (else            (aux (cdr l) (cons (car l) res))))))
    (aux l '())))

;==============================================================================
;                                     EXO 3
;==============================================================================
(define milieu
  (lambda (l)
    (if (or (null? l) (null? (cdr l)))
        '()
        (reverse (cdr (reverse (cdr l)))))))

#|
(milieu '(bonjour tout le monde))
(milieu '(hello world))
(milieu '(seul))
(milieu '(x (y) z))
(milieu '(a b))
(milieu '())
```



```
|#

;==============================================================================
;                                     EXO 4
;==============================================================================
; Deux versions utilisant chacune des fonctions définies dans les exercices
; précédents

(define swap1
  (lambda (l)
    (cons (car (reverse l))
          (append (milieu l)
                  (list (car l))))))

(define swap2
  (lambda (l)
    (append (rotation-droite (cdr l))
            (list (car l)))))
#|
(swap1 '(1 2))
(swap1 '(1 2 3 4))
(swap1 '((1 2 )(3 4)))
(swap1 '(x () y))
(swap1 '(1))
|#

;==============================================================================
;                                     EXO 5
;==============================================================================

; Les fonctions définies ci-dessous n'effectuent aucun test sur leurs
; arguments. Une écriture plus réaliste contiendrait de tels tests,
; afin de s'assurer que les arguments sont du bon type.
;
; L'ensemble de ces fonctions sont décrites dans le R^4RS, pages 16 et 17.

; LIST?
(define list?
  (lambda (l)
    (or (null? l)
        (and (pair? l) (list? (cdr l))))))

; LENGTH
(define length
  (lambda (l)
    (cond
      ((null? l) 0)
      ((pair? l) (+ 1 (length (cdr l))))
      (else      (error "Mauvaise liste")))))

; APPEND
(define append
  (lambda (a b)
    (cond
      ((null? a) b)
      ((pair? a) (cons (car a)
                       (append (cdr a) b)))
      (else      (error "Mauvaise liste")))))

; LIST-REF (Cette version ne marche que si la liste est bien formée)
(define _list-ref
  (lambda (l k)
```

```
    (if (zero? k)
        (car l)
        (list-ref (cdr l) (- k 1)))))

; MEMBER (Cette version ne marche que si la liste est bien formée)
(define member
  (lambda (x l)
    (cond
       ((null? l)           #f)
       ((equal? x (car l)) l)
       (else                (member x (cdr l))))))

;==============================================================================
;                                       EXO 6
;==============================================================================
(define entiers
  (lambda (l)
    (cond
     ((integer? l )1)
     ((pair? l)
       (+ (entiers (car l)) (entiers (cdr l))))
     (else 0))))
#|
(entiers '(1 (2 (34) 5) 6 () (78)))
(entiers '((((1 a b))) (((x 2)))))
(entiers '12)
(entiers '())
|#

;==============================================================================
;                                       EXO 7
;==============================================================================
(define reverse
  (lambda (l)
    (define aux (lambda (l res)
                  (cond
                     ((null? l) res)
                     ((pair? l) (aux (cdr l) (cons (car l) res)))
                     (else      (error "Mauvaise liste")))))
    (aux l '())))


(define full-reverse
  (lambda (l)
    (define aux (lambda (l res)
                  (cond
                   ((null? l) res)
                   ((pair? l) (aux (cdr l) (cons (full-reverse (car l))
                                                 res)))
                   (else l))))  ;; Attention ici on renvoie l et pas erreur...
    (aux l '())))

#|
(full-reverse '(1 (2 (3 4) 5)(6 7) (8)))
(full-reverse '(1 (2 (3 (4 (5))))))
(full-reverse 'symbole)
|#

;==============================================================================
;                                       EXO 8
;==============================================================================
(define flat
```



```
  (lambda (l)
    (cond
       ((null? l) l)
       ((pair? l) (append (flat (car l))
                          (flat (cdr l))))
       (else      (list l)))))

#|
(flat '(a(b (c)((d)))e()(f)))
(flat '(() 1()2()3))
(flat '(((((x))))))
(flat '(()()()()))
|#

;==============================================================================
;                                       EXO 9
;==============================================================================
(define is-in
  (lambda (x l)
    (cond ((null? l) #f)
          ((pair? l) (or (equal? x (car l))
                         (is-in x (car l))
                         (is-in x (cdr l))))
          (else #f))))

#|
(is-in 'y '(x (y z) w))
(is-in '(a b) '(x (y (a b) z) w))
(is-in '(a b) '(x a b y))
(is-in 'x '(x))
(is-in 'x '())
(is-in '() '())
|#
```