```
;;;;
;;;; Corrigé.stk        -- Corrigés de la feuille 5
;;;;
;;;;           Author: Erick Gallesio [eg@unice.fr]
;;;;    Creation date: 28-Feb-2000 09:06 (eg)
;;;; Last file update: 28-May-2003 11:59 (eg)
;;;;


;=============================================================================
;                                    EXO 1
;=============================================================================
(define (creer-additionneur k)
  (let ((c k))
    (lambda (n) (+ n c))))

(define add1 (creer-additionneur 1))
(define add5 (creer-additionneur 5))

#|
(add1 10)
(add5 10)
|#

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

(define (creer-accumulateur k)
  (let ((n k))
    (lambda (x)
      (set! n (+ x n))
      n)))

;;Autre version (sans let)
;; En effet, le let n'est pas obligatoire car on peut capturer le
;; paramètre de la fonction
(define (creer-accumulateur n)
  (lambda (x)
    (set! n (+ x n))
    n))

#|
(define a (creer-accumulateur 100))
(a 10)
(a 20)
(a 30)
|#

;=============================================================================
;                                    EXO 3
;=============================================================================

(define msg-special (cons 'msg 'special))   ;; Objet unique en mémoire
                                            ;; (car on fait cons)

(define (remember f)
  (let ((result #f))
    (lambda l
      (unless (and (pair? l) (eq? (car l) msg-special)) ;;  eq? et pas equal?
        (set! result (apply f l)))
      result)))
```



```
(define (last f)
  (f msg-special))


#|
(define biba (remember (lambda (x) (+ x 2))))
(define beloula (remember (lambda (x) (* x 3))))

(biba 10)
(last biba)
(beloula 10)
(biba 20)
(last biba)
(last beloula)
(beloula 100)
(last beloula)
|#

;=============================================================================
;                                    EXO 4
;=============================================================================

(define (memo f)
  (let ((a-list '()))
    (lambda l
      (let ((previous (assoc l a-list)))
        (if previous
            (cdr previous)
            (let ((res (apply f l)))
              (set! a-list (cons (cons l res) a-list))
              res))))))

#|
(define  (fib n) (if (< n 2) 1 (+ (fib(- n 1))(fib (- n 2)))))
(define mfib (memo fib))
(mfib 25)

;; Encore plus fort !!!!!!!
(set! fib (memo fib))

(fib 1000) ;; et c'est instantané!! Ce qui est important de voir ici c'est
           ;; qu'on a rendu une fonction "impraticable" en une fonction
           ;; utilisable pour de grandes valeurs de n. Ce qu'on a fait, c'est
           ;; qu'on a changé les règles d'évaluation d'une fonction (i.e. si le
           ;; résultat est connu on ne le reconstruit pas, on utilise la valeur
           ;; que l'on a retenue). Quant-à la fonction de l'utilisateur (fib ici
)
           ;; elle n'a pas été changée (i.e. c'est toujours la belle fonction
           ;; avec ses deux appels récursifs).
|#
;=============================================================================
;                                    EXO 5
;=============================================================================

;;; Version 1

(define (profiler f)
  (let ((count 0))
    (lambda l
      (if (equal? l '(stats))
          ;; Cas spécial où l'on affiche le compteur
```

```
          count
          ;; Cas normal
          (begin
            (set! count (+ count 1))
            (apply f l))))))

;;; Version 2
;;; La ruse consiste ici à capturer les fonctions dont on a besoin
;;; pour ne pas se mettre à boucler si on se on décide de les profiler
;;; (on conserve ainsi une copie locale des fonctions standard)

(define (profiler f)
  (let ((count 0)
        (+ +)(not not)(null? null?)(eqv? eqv?)(car car)(set! set!)(apply apply))

    (lambda l
      (if (equal? l '(stats))
          ;; Cas spécial où l'on affiche le compteur
          count
          ;; Cas normal
          (begin
            (set! count (+ count 1))
            (apply f l))))))


#|
(define my-sqrt (profiler sqrt))
(my-sqrt 100)
(my-sqrt 2)
(my-sqrt 'stats)

(define + (profiler +))
(+ 1 2 3)
(+ 'stats)

|#
```