Лабораторная работа № 3. Типы данных. Модульное тестирование

15 декабря 2023 г.

Бойко Роман, ИУ9-12Б

Цели работы

- На практике ознакомиться с системой типов языка Scheme.
- На практике ознакомиться с юнит-тестированием.
- Разработать свои средства отладки программ на языке Scheme.
- На практике ознакомиться со средствами метапрограммирования языка Scheme.

Реализация

```
;; Number 1
(define-syntax trace-ex
  (syntax-rules ()
    ((_ expr)
     (begin
       (write 'expr)
       (display " => ")
       (let ((x expr))
         (write x)
         (newline)
         x)))))


;; Number 2
(define-syntax test
  (syntax-rules ()
    ((test (func . param) expected)
     (list (func . param) expected '(func . param)))))

(define (run-test the-test)
  (if (equal? (car the-test) (cadr the-test))
```

```
      (begin
        (write (caddr the-test))
        (display " ok\n")
        #t)
      (begin
        (write (caddr the-test))
        (display " FAIL\n")
        (display "  Expected: ")
        (write (cadr the-test))
        (newline)
        (display "  Returned: ")
        (write (car the-test))
        (newline)
        #f)))

(define (run-tests the-tests)
  (let loop ((the-tests the-tests)
             (all-tests-passed #t))
    (cond
      ((null? the-tests) all-tests-passed)
      ((and (run-test (car the-tests)) all-tests-passed) (loop (cdr the-tests) #t))
      (else (loop (cdr the-tests) #f)))))


;; Number 3
(define (ref var ind . xs)
  (if (null? xs)
      (cond
        ((list? var)
         (if (< -1 ind (length var))
             (list-ref var ind)
             #f))
        ((string? var)
         (if (< -1 ind (string-length var))
             (string-ref var ind)
             #f))
        ((vector? var)
         (if (< -1 ind (vector-length var))
             (vector-ref var ind)
             #f)))
      (cond
        ((list? var) (insert-at var ind (car xs)))
        ((string? var)
         (if (char? (car xs))
             (begin
               (set! var (insert-at (string->list var) ind (car xs)) )
```

```
                (if var
                    (list->string var)
                    #f))
              #f))
         ((vector? var)
          (begin
            (set! var (insert-at (vector->list var) ind (car xs)))
            (if var
                (list->vector var)
                #f))))))

(define (insert-at lst index value)
  (if (or (< index 0) (> index (length lst)))
      #f
      (cond
        ((null? lst) (list value))
        ((= 0 index) (cons value lst))
        (else (cons (car lst) (insert-at (cdr lst) (- index 1) value ))))))


;; Number 4
(define (x xs)
  (cadadr xs))
(define (y ys)
  (car (cdaddr ys)))

(define (factorize expr)
  (cond
    ((and (equal? (car expr) '-) (equal? (car (cddadr expr)) 2))
     `(* (- ,(x expr) ,(y expr))
         (+ ,(x expr) ,(y expr))))
    ((and (equal? (car expr) '-) (equal? (car (cddadr expr)) 3))
     `(* (- ,(x expr) ,(y expr))
         (+ (* ,(x expr) ,(x expr))
            (* ,(x expr) ,(y expr))
            (* ,(y expr) ,(y expr)))))
    ((and (equal? (car expr) '+) (equal? (car (cddadr expr)) 3))
     `(* (+ ,(x expr) ,(y expr))
         (+  (* ,(x expr) ,(x expr))
             (- (* ,(x expr) ,(y expr)))
             (* ,(y expr) ,(y expr)))))))
```

Тестирование

```
Test 3
```

```
(define the-tests-ref
  (list (test (ref '(1 2 3) 1) 2) ;; ok
        (test (ref '(1 2 3) 0) 1) ;; ok
        (test (ref '(1 2 3) 3) #f) ;; index out of range
        (test (ref '(1 2 3) -1) #f) ;; index out of range

        (test (ref #(1 2 3) 1) 2) ;; ok
        (test (ref #(1 2 3) 0) 1) ;; ok
        (test (ref #(1 2 3) 3) #f) ;; index out of range
        (test (ref #(1 2 3) -1) #f) ;; index out of range

        (test (ref "123" 1) #\2) ;; ok
        (test (ref "123" 0) #\1) ;; ok
        (test (ref "123" -1) #f) ;; index out of range
        (test (ref "123" 3) #f) ;; index out of range

        (test (ref '(1 2 3) 1 0) '(1 0 2 3)) ;; ok
        (test (ref '(1 2 3) 1 "1") '(1 "1" 2 3)) ;; ok
        (test (ref '(1 2 3) 1 '(4 5)) '(1 (4 5) 2 3)) ;; ok
        (test (ref '(1 2 3) 1 #\r) '(1 #\r 2 3)) ;; ok
        (test (ref '(1 2 3) 1 #(4 5)) '(1 #(4 5) 2 3)) ;; ok
        (test (ref '(1 2 3) 5 0) #f) ;; index out of range
        (test (ref '(1 2 3) -1 0) #f) ;; index out of range

        (test (ref #(1 2 3) 1 #\0) #(1 #\0 2 3)) ;; ok
        (test (ref #(1 2 3) 1 "1") #(1 "1" 2 3)) ;; ok
        (test (ref #(1 2 3) 1 0) #(1 0 2 3)) ;; ok
        (test (ref #(1 2 3) 1 #(4 5)) #(1 #(4 5) 2 3)) ;; ok
        (test (ref #(1 2 3) 5 0) #f) ;; index out of range
        (test (ref #(1 2 3) -1 0) #f) ;; index out of range

        (test (ref "123" 3 #\4) "1234") ;; ok
        (test (ref "123" 1 #\4) "1423") ;; ok
        (test (ref "123" 1 "1") #f) ;; wrong type(string)
        (test (ref "123" 1 '(4 5)) #f) ;; wrong type(list)
        (test (ref "123" 1 0) #f) ;; wrong type(integer)
        (test (ref "123" 1 #(4 5)) #f) ;; wrong type(vector)
        (test (ref "123" 5 #\4) #f) ;; index out of range
        (test (ref "123" -1 #\4) #f) ;; index out of range
        ))

(run-tests the-tests-ref)

Test 4
(define the-tests-factorize
 (list (test (factorize '(- (expt x 2) (expt y 2)))                ;; a^2 -
```

```
b^2
             '(* (- x y) (+ x y)))
           (test (factorize '(- (expt (+ first 1) 2) (expt (-
second 1) 2)))     ;; a^2 - b^2 (complex expr)
              '(* (- (+ first 1) (- second 1)) (+ (+ first 1) (-
second 1))))
     (test (eval (list (list 'lambda '(x y)                    ;; a^2 -
b^2 with eval
                           (factorize '(- (expt x 2) (expt y 2))))
                         1 2) (interaction-environment))
             -3)
     (test (factorize '(- (expt x 3) (expt y 3)))                ;; a^3 -
b^3
             '(* (- x y) (+ (* x x) (* x y) (* y y))))
           (test (factorize '(- (expt (+ first 1) 3) (expt (-
second 1) 3)))     ;; a^3 - b^3 (complex expr)
             '(* (- (+ first 1) (- second 1))
                 (+ (* (+ first 1) (+ first 1))
                    (* (+ first 1) (- second 1))
                    (* (- second 1) (- second 1)))))
     (test (factorize '(+ (expt x 3) (expt y 3)))                ;; a^3 + b^3
             '(* (+ x y) (+ (* x x) (- (* x y)) (* y y))))
           (test (factorize '(+ (expt (+ first 1) 3) (expt (-
second 1) 3)))     ;; a^3 + b^3 (complex expr)
             '(* (+ (+ first 1) (- second 1))
                 (+ (* (+ first 1) (+ first 1))
                    (- (* (+ first 1) (- second 1)))
                    (* (- second 1) (- second 1)))))
     (test (factorize '(+ (expt x 5) (expt y 5)))                ;; wrong parametrs
             '(+ (expt x 5) (expt y 5)))
       ))

(run-tests the-tests-factorize)
```

Вывод

Ознакомился с системой типов языка Scheme, с юнит-тестированием. Разработал свои средства отладки программ на языке Scheme. Ознакомился со средствами метапрограммирования языка Scheme.