Лабораторная работа № 3. Типы данных. Модульное тестирование

16 декабря 2023 г.

Дмитрий Лимонов, 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

```
(define-syntax trace-ex
  (syntax-rules ()
    ((trace-ex smth) (begin
                       (let ((x smth))
                         (write 'smth)
                         (display " => ")
                         (write x)
                         (display "\n")
                         x)))))
(define (zip . xss)
  (if (or (null? xss)
          (null? (trace-ex (car xss))))
      '()
      (cons (map car xss)
            (apply zip (map cdr (trace-ex xss))))))

(define-syntax test
  (syntax-rules ()
    ((test tes ans) '(tes ans))))

(define (run-test testik)
  (write (car testik))
  (define c (eval (car testik) (interaction-environment)))
  (if (equal? c (cadr testik))
      (begin
```

```
        (display " ok\n")
        #t)
      (begin
        (display " FAIL\n")
        (display "  Expected: ")
        (write (cadr testik))
        (display " \n")
        (display "  Returned: ")
        (write c)
        (display "\n")
        #f)))

(define (run-tests sp)
  (if (null? (cdr sp))
      (run-test (car sp))
      ((lambda (x y)
         (and x y))
       (run-test (car sp))
       (run-tests (cdr sp)))))

(define (insert sp index elem)
  (if (= 0 index)
      (cons elem (insert sp (- index 1) elem))
      (if (= 0 (length sp))
          '()
          (cons (car sp) (insert (cdr sp) (- index 1) elem)))))


(define (ref elem . xs)
  (if (= (length xs) 1)
      (let* ((ind (car xs)))
        (cond
          ((string? elem) (and (not (<= (string-length elem) ind)) (string-ref elem ind)))
          ((list? elem)  (and (not (<= (length elem) ind)) (list-ref elem ind)))
          ((vector? elem) (and (not (<= (vector-length elem) ind)) (vector-ref elem ind)))))
      (let* ((ind (car xs))
             (exch (cadr xs)))
        (define t (cond
                    ((string? elem) (list (string->list elem) "str"))
                    ((vector? elem) (list (vector->list elem) "vec"))
                    ((list? elem) (list elem "lis"))))
        (and (>= (length (car t)) ind)
             (begin
               (let ((ans (insert (car t) ind exch)))
                 (cond
                   ((equal? (cadr t) "str") (and (char? exch) (list->string ans)))
```

```
                ((equal? (cadr t) "vec") (list->vector ans))
                ((equal? (cadr t) "lis") ans))))))
      ))

(define (factorize sp)
  (let* ((x (cadr (cadr sp)))
         (y (cadr (caddr sp)))
         (li sp))
    (if (= 2 (caddr (cadr li)))
        `(* (- x y) (+ x y))
        (if (equal? '+ (car li))
            `(* (+ x y) (- ( + (* x x) (* y y)) (* x y)))
            `(* (- x y) (+ (* x x) (* y y) (* x y)))
            ))))
```

Тестирование

```
Welcome to DrRacket, version 8.11 [cs].
Language: R5RS; memory limit: 128 MB.
> (zip '(1 2 3) '(one two three))
(car xss) => (1 2 3)
xss => ((1 2 3) (one two three))
(car xss) => (2 3)
xss => ((2 3) (two three))
(car xss) => (3)
xss => ((3) (three))
(car xss) => ()
((1 one) (2 two) (3 three))
> (define (signum x)
  (cond
    ((< x 0) -1)
    ((= x 0)  1) ; Ошибка здесь!
    (else     1)))
> (define the-tests
  (list (test (signum -2) -1)
        (test (signum  0)  0)
        (test (signum  2)  1)))
> (run-tests the-tests)
(signum -2) ok
(signum 0) FAIL
  Expected: 0
  Returned: 1
(signum 2) ok
#f
> (ref '(1 2 3) 1)
```

```
(ref #(1 2 3) 1)
(ref "123" 1)
(ref "123" 3)
(ref '(1 2 3) 1 0)
(ref #(1 2 3) 1 0)
(ref #(1 2 3) 1 #\0)
(ref "123" 1 #\0)
(ref "123" 1 0)
(ref "123" 3 #\4)
(ref "123" 5 #\4)
2
2
#\2
#f
(1 0 2 3)
#(1 0 2 3)
#(1 #\0 2 3)
"1023"
#f
"1234"
#f
> (factorize '(- (expt x 2) (expt y 2)))
(* (- x y) (+ x y))
> (factorize '(- (expt (+ first 1) 2) (expt (- second 1) 2)))
(* (- (+ first 1) (- second 1)) (+ (+ first 1) (- second 1)))
> (eval (list (list 'lambda
                    '(x y)
                    (factorize '(- (expt x 2) (expt y 2))))
              1 2)
         (interaction-environment))
-3
```

Вывод

Был создан один из мощнейших инструментов поисков ошибок в программе - макрос trace-ex, что позволяет упростить разработку программ в будущем. Также создан каркас для юнит-тестирования, что также является очень важным элементом отладки готового продукта.