# Лабораторная работа № 4. Метапрограммирование. Отложенные вычисления

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 call/cc call-with-current-continuation)

(define (void? x)
  (equal? x (if #f #f)))


(define moment #f)
(define (use-assertions)
  (define err (call/cc
               (lambda (cc)
                 (set! moment cc))))
  (if (not (void? err))
      (begin
        (display "FAILED: ")
        (write err))))

(define-syntax assert
  (syntax-rules ()
    ((assert expr) (let ((ans (eval expr (interaction-environment))))
                     (or ans (moment 'expr))))
    ))

;;;;;;;;;;
```

```
(define (save-data data path)
  (define out (open-output-file path))
  (write data out)
  (close-output-port out))

(define (load-data path)
  (call-with-input-file path
    (lambda (port)
      (read port))))

(define (work file)
  (let ((obj (read file)))
    (if (eof-object? obj)
        0
        (if (equal? obj "\n")
            (work file)
            (+ 1 (work file))))))

(define (count path)
  (define in (open-input-file path))
  (work in))

;;;;;;;;;;

(define trib_memo
  (let ((known-results '()))
    (lambda (num)
      (cond
        ((<= num 1) 0)
        ((= num 2) 1)
        (else
         (let* ((res (assoc num known-results)))
           (if res
               (cadr res)
               (begin
                 (set! res (+ (trib_memo (- num 1)) (trib_memo (- num 2)) (trib_memo (- num
                 (set! known-results (cons (list num res) known-results))
                 res))))))))))

;;;;;;;;;;

(define-syntax my-if
  (syntax-rules ()
    ((my-if correct truth lie) (force (or (and correct (delay truth)) (delay lie))))))
```

```
;;;;;;;;;;

(define-syntax my-let*
  (syntax-rules ()
    ((my-let ((a b) ...) action) (begin
                                   (define a b) ...
                                   action))
    ((my-let ((a b) ...) action . xs) (begin
                                        (define a b) ...
                                        action
                                        (my-let* ((a b) ...) . xs)))))

(define-syntax my-let
  (syntax-rules ()
    ((my-let ((a b) ...) action) ((lambda (a ...) action) b ...))
    ((my-let ((a b) ...) action . xs) (begin
                                        ((lambda (a ...) action) b ...)
                                        (my-let ((a b) ...) . xs)))))

;;;;;;;;;;

(define-syntax when
  (syntax-rules ()
    ((when cond action) (and cond action))
    ((when cond action . xs) (begin
                               (and cond action)
                               (when cond . xs)))))

(define-syntax unless
  (syntax-rules ()
    ((unless cond action) (and (not cond) action))
    ((unless cond action . xs) (begin
                                 (and (not cond) action)
                                 (unless cond . xs)))))

;;;;;;;;;;
```

## Тестирование

```
Welcome to DrRacket, version 8.11 [cs].
Language: R5RS; memory limit: 128 MB.
> (use-assertions)
> (define (1/x x)
  (assert (not (zero? x))) ; Утверждение: x ДОЛЖЕН БЫТЬ ≠ 0
  (/ 1 x))
```

```
> (map 1/x '(1 2 3 4 5)) ; ВЕРНЕТ список значений в программу

(map 1/x '(-2 -1 0 1 2)) ; ВЫВЕДЕТ в консоль сообщение и завершит работу программы
(1 1/2 1/3 1/4 1/5)
FAILED: (not (zero? x))
> (my-if #t 1 (/ 1 0))
(my-if #f (/ 1 0) 1)
1
1
> (define x 1)
> (when   (> x 0) (display "x > 0")  (newline))
(unless (= x 0) (display "x != 0") (newline))
x > 0
x != 0
```

## Вывод

Создание собственноручно встроенных конструкций языка позволяет углубить знания об этом языке. Так, создание макросов my-let и my-let* раскрывает новый инструмент - эллипсис, что позволяет упростить написание кода на языке Scheme. Также континуации очень дают возможность создать свои методы отлавливания ошибок внутри кода, что также упрощает разработку.