Разбор задач анкеты от 28 сентября

- С помощью функций высшего порядка map, foldl, foldr, filter опишите функцию (task28a lst) принимающую непустой список списков чисел. Функция находит среднее арифметическое максимумов модулей элементов вложенных списков.

```
(define (task22a lst)
  (* 1.0 (/ (foldl + 0 (map (lambda (ls)
                              (foldl max 0 (map abs ls))) lst))
            (length lst))))
```

Разбор задач анкеты от 28 сентября

- С помощью функций высшего порядка map, foldl, foldr, filter опишите функцию (task28b lst) принимающую непустой список списков чисел. Функция находит максимальное среди всех средних арифметических модулей элементов вложенных списков.

```
(define (task22b lst)
   (foldl max 0 (map (lambda (ls)
                       (if (null? ls) 0
                           (* 1.0 (/ (foldl + 0
                                            (map abs ls))
                                     (length ls)))
                       )) lst)))
```

ЛЕКЦИЯ 5
Остаточные вычисления. Стиль передачи остаточных вычислений

Остаточные вычисления

- Остаточные вычисления (continuations) -- что-то, что ждёт значение. С каждым промежуточным значением связано остаточное вычисление -- вычисления, которые должны быть выполнены, как только значение станет известным.
- Например: (sqrt (+ (read) 1)) ждёт, пока пользователь не введет число.
- Остаточные вычисления для функции foo в выражении (* (foo 1) (+ 5 1)) можно описать через lambda:
- (lambda (x) (* x (+ 5 1)))

Остаточные вычисления

- Остаточные вычисления – динамические объекты. На каждом шаге программы существуют текущие остаточные вычисления
- Например:

```
(define (fact n) (if (= n 0) 1 ( * n (fact (- n 1)))))

(fact 3)
 (* 3 (fact 2))
 (* 3 (* 2 (fact 1)))
 (* 3 (* 2 (* 1 (fact 0))))
…
=> 6
```

Факториал в стиле передачи о. в.

Перепишем факториал в виде функции с дополнительным параметром -- функцией, представляющей собой остаточные вычисления:

```
(define (fact-cc n cc)
    (if (= n 0) (cc 1)
        (fact-cc (- n 1) (lambda (x) (cc (* n x))))))
```

Пример вызова:

```
(fact-cc 2 (lambda (x) x))
(fact-cc 1 (lambda (x1) ((lambda (x) x) (* 2 x1))))
(fact-cc 0 (lambda (x2) ((lambda (x1)((lambda (x)x)(* 2 x1))) (* 1
    x2))))
((lambda (x2) ((lambda (x1) ((lambda (x) x) (* 2 x1))) (* 1 x2))) 1)
```

Факториал в стиле передачи о. в. Продолжение

((lambda (x_2) ((lambda (x_1) ((lambda (x) x) (* 2 x_1))) (* 1 x_2))) 1)

((lambda (x_1) ((lambda (x) x) (* 2 x_1))) (* 1 1))

((lambda (x) x) (* 2 1))

2

reverse в стиле передачи о. в.

- Опишем reverse в continuation-passing style
- Сначала определим схему вычислений:
  - Результат можно получить, вставляя элементы исходного списка по одному от начала к хвосту в голову списка-результата.
  - reverse в обычном стиле:

```
(define (reverse-simple lst)
    (define (loop l res)
            (if (null? l) res (loop (cdr l) (cons (car l) res))))
    (loop lst '()))
```

  - пример: (loop (list 1 2 3) '())
    (loop (list 2 3) (cons 1 '()))
    (loop (list 3) (cons 2 (cons 1 '()))) …

reverse в стиле передачи о. в.

- Пишем в continuation-passing style

```
(define (reverse-cps lst cc)
        (if (null? lst) (cc '())
            (reverse-cps (cdr lst) (lambda (x) (cons (car lst) (cc x))))))
```

- что общего с обычным описанием?

```
(define (loop l res)
        (if (null? l) res (loop (cdr l) (cons (car l) res))))
```

- пример работы

(reverse-cps '(1 2) (lambda (x) x))

(reverse-cps '(2) (lambda (x_1) (cons 1 ((lambda (x) x) x_1))))

(reverse-cps '() (lambda (x_2) (cons 2 ((lambda (x_1) (cons 1 ((lambda (x) x) x_1))) x_2))))

reverse в стиле передачи о. в.

(reverse-cps '() (lambda (x_2) (cons 2 ((lambda (x_1) (cons 1 ((lambda (x) x) x_1))) x_2))))

((lambda (x_2) (cons 2 ((lambda (x_1) (cons 1 ((lambda (x) x) x_1))) x_2))) '())

(cons 2 ((lambda (x_1) (cons 1 ((lambda (x) x) x_1))) '()))

(cons 2 (cons 1 ((lambda (x) x) '())))

(cons 2 (cons 1 '()))

(cons 2 '(1))

'(2 1)

Вставка в хвост в стиле передачи о. в.

- Опишем app-cps, добавляющую элемент в хвост списка:
- Сначала определим схему вычислений:
  - Результат можно получить, приписав голову исходного списка к результату вставки в хвост (рекурсивной). Вставка в пустой список получается как список из нового элемента.
  - пишем в обычном стиле:

```
(define (app-simple lst e)
    (if (null? lst) (list e)
                (cons (car lst) (app-simple (cdr lst) e))))
```

  - пример работы

```
(app-simple ‘(1 2) 3)
(cons 1 (app-simple ‘(2) 3))
(cons 1 (cons 2 (app-simple ‘() 3))) …
```

Вставка в хвост в стиле передачи о. в.

- Опишем app-cps, добавляющую элемент в хвост списка:

```
(define (app-cps lst e cc)
    (if (null? lst) (cc (list e))
    (app-cps (cdr lst) e (lambda (k) (cc (cons (car lst) k))))))
```

- Что общего с обычной версией?

```
(define (app-simple lst e)
    (if (null? lst) (list e)
            (cons (car lst) (app-simple (cdr lst) e))))
```

- Пример:

(app-cps ‘(1 2) 3 (lambda (x) x))

(app-cps ‘(2) 3 (lambda (x_1) ((lambda (x) x) (cons 1 x_1))))

(app-cps ‘() 3 (lambda (x_2) ((lambda (x_1) ((lambda (x) x) (cons 1 x_1))) (cons 2 x_2))))

Вставка в хвост в стиле передачи о. в.

(app-cps ‘() 3 (lambda (x_2) ((lambda (x_1) ((lambda (x) x) (cons 1 x_1))) (cons 2 x_2))))

((lambda (x_2) ((lambda (x_1) ((lambda (x) x) (cons 1 x_1))) (cons 2 x_2))) '(3))

(((lambda (x_1) ((lambda (x) x) (cons 1 x_1))) (cons 2 ‘(3)))

((lambda (x) x) (cons 1 (cons 2 ‘(3))))

(cons 1 (cons 2 ‘(3)))

(cons 1 ‘(2 3))

‘(1 2 3)

Итоги лекции 5

- Преобразуя программу согласно cps мы достигаем двух целей:
  - Всю рекурсию делаем хвостовой.
  - В интерпретаторе, вычисляющем в аппликативном порядке, реализуем вычисления в нормальном порядке.

Немного повторения перед анкетой

- length-cps
- сначала обычный

```
(define (length-simple lst)
    (if (null? lst) 0
        (+ 1 (length-simple (cdr lst)))))
```

- пример

```
(length-simple ‘(1 2 3))
(+ 1 (length-simple ‘(2 3)))
(+ 1 (+ 1 (length-simple ‘(3))))
(+ 1 (+ 1 (+ 1 (length-simple ‘()))))
(+ 1 (+ 1 (+ 1 0)))
(+ 1 (+ 1 1))
(+ 1 2)          3
```

Немного повторения перед анкетой

- готовы писать length-cps

```
(define (length-cps lst cc)
  (if (null? lst) (cc 0)
      (length-cps (cdr lst) (lambda (x) (+ 1 (cc x))))))
```