Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 10
Рекурсивные функции

Студент Лучина Е.Д

Группа ИУ7-61Б

Преподаватель Толпинская Н.Б.

Москва.

2020 г.

7. Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list, т.е. например для аргумента ((1 2) (3 4)) -> 4.

```
defun cnt_len (lst)
    (cond
        ((null lst) 0)
        ((listp (car lst)) (+ (cnt_len (car lst)) (cnt_len (cdr lst))))
        (t (+ 1 (cnt_len (cdr lst))))
    )
)

(print (cnt_len  '(1 2 (3 4) 2))) -> 5
```

Если список пуст, результат - 0

Иначе если первый элемент - список, результат равен сумме длины этого элемента и хвоста

Иначе результат - длина хвоста, увеличенная на 1

```
(defun cnt_len2 (lst)
    (reduce #'+ (mapcar (lambda (x) (if (listp x) (cnt_len2 x) 1)) lst))
)

(print (cnt_len2  '(1 2 (3 (8) 4) 2))) -> 6

(1 2 (3 (8) 4) 2) -> (1 1 (3 (8) 4) 2) -> (1 1 (1 (8) 4) 2) ->
(1 1 (1 (1) 4) 2) -> (1 1 (1 1 4) 2) -> (1 1 (1 1 1) 2) ->
(1 1 (2 1) 2) -> (1 1 3 2) -> (1 1 3 1) -> (2 3 1) -> (5 1) -> 6
```

8. Написать рекурсивную версию (с именем reg-add) вычисления суммы чисел заданного списка.

Например: (reg-add (2 4 6)) -> 12

```
(defun reg_add (lst)
    (cond
        ((null lst) 0)
        (t (+ (car lst) (reg_add (cdr lst))))
    )
)
```

Если список пустой результат равен 0

Иначе сумме значения этого элемента и сумме последующих

9. Написать рекурсивную версию с именем recnth функции nth.

```
(defun recnth (n lst)
```

```
    (cond
        ((null lst) nil)
        ((eq n 0) (car lst))
        (t (recnth (- n 1) (cdr lst)))
    )
)

(print (recnth 1 '(1 3 4))) -> 3
```

10. Написать рекурсивную функцию alloddr, которая возвращает t когда все

элементы списка нечетные.

```
(defun alloddr (lst)
  (cond ((null lst) t)
        ((evenp (car lst)) nil)
        ( t (alloddr (cdr lst)))
   )
)

(alloddr ‘(1 3 5)) -> T
(alloddr ‘()) -> T
(alloddr ‘(1 2)) -> Nil
```

11. Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка - аргументы.

```
(defun ends (lst)
    (cond ((null lst) nil)
        ((null (cdr lst)) (car lst))
        (t (ends (cdr lst)))
     )
)
```

12. Написать рекурсивную функцию, относящуюся к дополняемой рекурсии с одним тестом завершения, которая вычисляет сумму всех чисел от 0 до n-ого аргумента функции.

```
(defun get_sum (n lst)
    (cond ((null lst) 0)
        ((eq n 0) 0)
        ((numberp (car lst))
            (+ (car lst) (get_sum (- n 1) (cdr lst))))
        (t (get_sum (- n 1) (cdr lst)))
    )
)
```

```
(defun func (n &rest args)
    (get_sum n args)
)

(print (func 3 3 'd 4 6 7 8)) -> 7
N = 3, lst = (3 d 4 6 7 8); 3 + 4 = 7
```

1) от n-аргумента функции до последнего

```
(defun get_sum (lst)
    (cond ((null lst) 0)
        ((numberp (car lst)) (+ (car lst) (get_sum (cdr lst))))
        (t (get_sum (cdr lst)))
    )
)

(defun skip (n lst)
    (cond ((null lst) nil)
        ((eq n 0) lst)
        (t (skip (- n 1) (cdr lst)))
    )
)

(defun func (n &rest args)
    (get_sum (skip n args))
)

(func 1 5 4 3 1) -> 8
```

2) от n-аргумента функции до т-аргумента с шагом d.

```
(defun skip (n lst)
    (cond ((null lst) nil)
        ((eq n 0) lst)
        (t (skip (- n 1) (cdr lst)))
    )
)

(defun get_sum (n d lst)
    (cond ((null lst) 0)
        ((or (eq n 0) (< n 0)) 0)
        ((numberp (car lst)) (+ (car lst) (get_sum (- n d) d (skip d
lst))))
        (t (get_sum (- n d) d (skip d lst)))
    )
)

(defun func (s e d &rest args)
    (get_sum (- e s) d (skip s args))
)
```

(func 1 4 2 3 2 5 4 1); список (3 2 5 4 1) сумма каждого второго элемента в диапазоне от 1 до 4. -> 2 + 4 -> 6

Если элемент не число, оно пропускается (его значение не добавляется в результирующую сумму)

13. Написать рекурсивную функцию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции.

в num хранится последнее найденное нечетное число

```
(defun newodd (num lst)
    (cond
        ((null lst) num)
        ((oddp (car lst)) (newodd (car lst) (cdr lst)))
        (t (newodd num (cdr lst)))
    )
)

(defun lastodd (lst)
    (newodd nil lst)
)

(lastodd '(4 1 2 5 6 8)) -> 5
```

14. Используя cons-дополняемую рекурсию с одним тестом завершения, написать функцию которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.

```
(defun to2 (lst)
    (cond
        ((null lst) nil)
        (t (cons (*(car lst) (car lst)) (to2 (cdr lst))))
    )
)

(to2 '(1 2 3 4)) -> (1 4 9 16)
```

15. Написать функцию с именем select-odd, которая из заданного

списка выбирает все нечетные числа.

```
(defun select-odd (lst)
    (cond ((null lst) nil)
        ((and (numberp (car lst)) (oddp (car lst)))
            (cons (car lst) (select-odd (cdr lst))))
        (t (select-odd (cdr lst)))
```

```
    )
)

(print (select-odd '(1 2 3 s 4 (5) 61 7))) -> (1 3 61 7)
```

Вариант 1: select-even

```
(defun select-even (lst)
    (cond ((null lst) nil)
        ((and (numberp (car lst)) (evenp (car lst))) (cons (car lst)
(select-even (cdr lst))))
        (t (select-even (cdr lst)))
    )
)

(select-even '(1 2 3 s 4 (5) 61 7)) -> (2 4)
```

вариант 2: вычисляет сумму всех нечетных чисел(sum-all-odd) или сумму всех четных чисел (sum-all-even) из заданного списка.

```
(defun select-even (lst)
    (cond ((null lst) nil)
        ((and (numberp (car lst)) (evenp (car lst))) (cons (car lst)
(select-even (cdr lst))))
        (t (select-even (cdr lst)))
    )
)

(defun get_sum (lst)
    (cond ((null lst) 0)
        ((numberp (car lst)) (+ (car lst) (get_sum (cdr lst))))
        (t (get_sum (cdr lst)))
    )
)

(defun sum-all-even (lst)
    (get_sum (select-even lst))
)

(select-even '(1 2 3 s 4 (5) 61 7)) -> 6
```

Теоретические вопросы:

· Способы организации повторных вычислений в Lisp

- Использование функционалов (функция, которая принимает другую функцию в качестве параметра)
- Использование рекурсии (ссылка на себя)

· Что такое рекурсия? Способы организации рекурсивных функций

Рекурсия — это ссылка на определяемый объект во время его определения.

Существуют следующие типы рекурсивных функций:

- Хвостовая рекурсия
- Дополняемая рекурсия
- Множественная рекурсия
- взаимная рекурсия
- рекурсия более высокого порядка

· Различные способы организации рекурсивных функций и порядок их реализации

- хвостовая рекурсия

формирование результата не на выходе из рекурсии, а на входе в рекурсию, выполнение всех действий до ухода на следующий шаг рекурсии

```
(defun fun (x)
      (cond (end_test1  end_value1)
                  ...
            (end_testN  end_valueN)
            ( t         (fun reduced_x) )
      ) )
```

- дополняемая рекурсия

при обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова , а вне его

```
(defun fun (x)
      (cond (test end_value)
            (t (add_fun add_value   (fun reduced_x)) )
))
```

- множественная рекурсия

На одной ветке происходит сразу несколько рекурсивных вызовов.

```
(defun fun (x)
      (cond (test end_val)
                  ( t (combine      (fun changed1_x)
                              (fun changed2_x))
                  )
))
```

- взаимная рекурсия

используется несколько функций, рекурсивно вызывающих друг друга

```
(defun fun2 (x)
      (cond (test end_val)
                  ( t (fun1 changed_x))
      )
))

(defun fun1 (x)
      (cond (test end_val)
                  ( t (fun2 changed_x))
      )
))
```

- рекурсия более высокого порядка

в теле определения функции аргументом рекурсивного вызова является рекурсивный вызов

```
(defun function_1 … (function_1 … (function_1 …) … ) … )
```

· Способы повышения эффективности реализации рекурсии

В целях повышения эффективности рекурсивных функций используется хвостовая рекурсия, суть которой в формировании результата не на выходе из рекурсии, а на входе в рекурсию, выполнении всех действий до ухода на следующий шаг рекурсии.

Преобразование не хвостовой рекурсии в хвостовую возможно путем использования дополнительных параметров. В этом случае необходимо использовать функцию-оболочку для запуска рекурсивной функции с начальными значениями дополнительных параметров