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: Луговой Д.М.
Группа: ИУ7-61Б
Преподаватель: Толпинская Н.Б.

Москва, 2020 г.

Цель работы: приобрести навыки организации рекурсии в Lisp.

Задачи работы: изучить способы организации хвостовой, дополняемой, множественной, взаимной рекурсии и рекурсии более высокого порядка в Lisp.

Лабораторная работа 6

Задание 7

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

С использованием функционалов:

Листинг 1: Функция вычисления суммарной длины списков

```
(defun count-length (list-of-lists)
  (reduce #'+
    (mapcar #'(lambda (x)
        (cond
          (
            (atom x)
            1
          )
          (
            (count-length x)
          )
        )
      ) list-of-lists
    )
  )
)
```

list-of-lists - входной список.

С помощью mapcar для каждого элемента проверяется, является ли он списком или атомом, если он является атомом, то его длина равна 1, иначе для него рекурсивно вызывается текущая функция. Затем с помощью reduce осуществляется суммирование длин элементов списка.

Хвостовая рекурсия:

Листинг 2: Функция-обертка для вычисления суммарной длины списков

```
(defun count-length (lst)
  (count-length-helper lst 0)
)
```

lst - входной список.

Листинг 3: Функция для вычисления суммарной длины списков

```
(defun count-length-helper (lst length)
 (cond
   (
     (null lst)
     length
   )
   (
     (listp (car lst))
     (count-length-helper
       (cdr lst)
       (count-length-helper (car lst) length))
     )
   (
     (count-length-helper (cdr lst) (+ length 1))
   )
 )
)
```

lst - входной список, **length** - длина списка.

Условием выхода из рекурсии является достижение конца списка(первый аргумент - nil) - возвращается второй аргумент, в котором накапливается суммарная длина списка. Если голова первого аргумента список, то осуществляется рекурсивный вызов текущей функции для хвоста первого аргумента и рекурсивного вызова, который осуществляется для головы первого аргумента и второго аргумента. Иначе осуществляется рекурсивный вызов текущей функции для хвоста списка и второго аргумента, увеличенного на 1.

Примеры работы:

```
> (count-length nil)
0
> (count-length '(1))
1
> (count-length '(1 2 3))
3
> (count-length '((1 2) (3 4)))
4
> (count-length '((1 2 (3) 4) 5 (6)))
6
```

Задание 8

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

Множественная рекурсия:

Листинг 4: Функция вычисления суммы чисел списка

```
(defun rec-add (lst)
  (cond
    (
      (numberp lst)
      lst
    )
    (
      (atom lst)
      0
    )
    (
      (+
        (rec-add (car lst))
        (rec-add (cdr lst))
      )
    )
  )
)
```

lst - входной список.

Условиями выхода из рекурсии являются:

- нахождение элемента, являющегося числом - возвращается этот элемент;
- нахождение элемента, являющегося атомом - возвращается 0.

Иначе осуществляются рекурсивные вызовы текущей функции для головы и хвоста аргумента, а результаты вызовов складываются и возвращаются.

Хвостовая рекурсия:

Листинг 5: Функция-обертка для вычисления суммы чисел списка

```
(defun rec-add (lst)
  (rec-add-helper lst 0)
)
```

lst - входной список.

Листинг 6: Функция вычисления суммы чисел списка

```
(defun rec-add-helper (lst sum)
  (cond
    (
      (null lst)
      sum
    )
    (
      (listp (car lst))
      (rec-add-helper (cdr lst) (rec-add-helper (car lst) sum)) )
```

```
10      (
11        (numberp (car lst))
12        (rec-add-helper (cdr lst) (+ sum (car lst)))
13      )
14      (
15        (rec-add-helper (cdr lst) sum)
16      )
17    )
18  )
```

lst - входной список, **sum** - сумма чисел списка.

Условием выхода из рекурсии является достижение конца списка (первый аргумент - nil) - возвращается второй аргумент, в котором накапливается сумма чисел списка. Иначе, если голова первого аргумента список, то осуществляется рекурсивный вызов текущей функции для хвоста первого аргумента и рекурсивного вызова, который осуществляется для головы первого аргумента и второго аргумента. Если голова первого аргумента - число, то осуществляется рекурсивный вызов текущей функции для хвоста списка и суммы второго аргумента и головы первого аргумента. Если голова первого аргумента не число и не список, то осуществляется рекурсивный вызов текущей функции для хвоста первого аргумента и второго аргумента.

Примеры работы:

```
1  > (rec-add '(1))
2  1
3  > (rec-add nil)
4  0
5  > (rec-add '(1 2 3))
6  6
7  > (rec-add '(1 2 (3 4) (5 (6))))
8  21
9  > (rec-add '(1 2 (a b (3)) nil (2 0)))
10 8
```

Задание 9

Написать рекурсивную версию с именем rec-nth функции nth.

Листинг 7: Рекурсивная версия функции nth

```
1  (defun rec_nth (lst n)
2    (cond
3      (
4        (null lst)
5        nil
6      )
7      (
8        (= n 0)
```

```
9         (car lst)
10      )
11      (
12        (rec_nth (cdr lst) (- n 1))
13      )
14    )
15  )
```

lst - входной список, **n** - индекс элемента, который нужно получить (начиная с 0).

Условиями выхода из рекурсии являются:

- достижение конца списка(первый аргумент функции - nil) - возвращается nil;
- нахождение искомого элемента списка (второй аргумент функции равен 0) - возвращается этот элемент.

Иначе осуществляется рекурсивный вызов текущей функции для хвоста первого аргумента и второго аргумента, уменьшенного на 1.

Примеры работы:

```
1 > (rec_nth nil 10)
2 NIL
3 > (rec_nth '(1) 0)
4 1
5 > (rec_nth '(1 (2 3) (4) 5 6) 1)
6 (2 3)
7 > (rec_nth '(1 2 3) 10)
8 NIL
```

Задание 10

Написать рекурсивную функцию alloddr, которая возвращает t, когда все элементы списка нечетные.

Множественная рекурсия:

Листинг 8: Функция проверки на нечетность всех элементов списка

```
1 (defun alloddr (lst)
2   (cond
3     (
4       (or
5         (null lst)
6         (and (numberp lst)(oddp lst))
7       )
8     )
```

```
    (
      (atom lst)
      nil
    )
    (
      (and
        (alloddr (car lst))
        (alloddr (cdr lst))
      )
    )
  )
)
```

lst - входной список.

Условиями выхода из рекурсии являются:

- достижение конца списка(аргумент функции - nil) или нахождение нечетного числа - возвращается t;
- нахождение элемента, вялящегося атомом, не подходящим по первому условию - возвращается nil.

Иначе осуществляются рекурсивные вызовы текущей функции для головы и хвоста аргумента, а возвращается логическое умножение этих вызовов.

Хвостовая рекурсия:

Листинг 9: Функция-обертка проверки на нечетность всех элементов списка

```
(defun alloddr (lst)
  (alloddr-helper lst t)
)
```

lst - входной список.

Листинг 10: Функция проверки на нечетность всех элементов списка

```
(defun alloddr-helper (lst isodd)
  (cond
    (
      (or (null lst)(not isodd))
      isodd
    )
    (
      (listp (car lst))
      (alloddr-helper
        (cdr lst)
        (alloddr-helper (car lst) isodd)
      )
    )
    (
      (and
        (numberp (car lst))
```

```
17          (oddp (car lst))
18        )
19        (alloddr-helper (cdr lst) isodd)
20      )
21    )
22  )
```

lst - входной список, **isodd** - логическая переменная для определения, содержаться ли в списке только нечетные числа, или нет.

По умолчанию считается, что список состоит только из нечетных чисел, поэтому начальное значение **isodd** - t.

Условием выхода из рекурсии является достижение конца списка(первый аргумент - nil) или нахождение элемента, являющего не нечетным(второй аргумент - nil) - возвращается второй аргумент. Если голова первого аргумента список, то осуществляется рекурсивный вызов текущей функции для хвоста первого аргумента и рекурсивного вызова, который осуществляется для головы первого аргумента и второго аргумента. Иначе, если голова первого аргумента - нечетное число, то осуществляется рекурсивный вызов текущей функции для хвоста списка и второго аргумента.

Примеры работы:

```
1  > (alloddr nil)
2  T
3  > (alloddr '(1))
4  T
5  > (alloddr '(1 2))
6  NIL
7  > (alloddr '(1 3))
8  T
9  > (alloddr '(1 (3 (5) 7) 9))
10 T
11 > (alloddr '(1 (3 nil) 2))
12 NIL
```

Задание 11

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

Листинг 11: Функция получения последнего элемента списка

```
1  (defun get_last (lst)
2    (cond
3      (
4        (null (cdr lst))
5        (car lst)
6      )
```

```
7       (
8         (get_last (cdr lst))
9       )
10    )
11  )
```

lst - входной список.

Условием выхода из рекурсии является достижение конца списка (хвост аргумента функции - nil) - возвращается голова аргумента. Иначе осуществляется рекурсивный вызов текущей функции для хвоста аргумента.

Примеры работы:

```
1 > (get_last nil)
2 NIL
3 > (get_last '(1))
4 1
5 > (get_last '(1 2 3 4 5))
6 5
7 > (get_last '(1 2 (3 4) (5) 6 7))
8 7
9 > (get_last '(1 2 (3 4)))
10 (3 4)
```

Задание 12

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

Листинг 12: Функция вычисления суммы элементов от 0 до n-го

```
1  (defun sum_to_n (lst n)
2    (cond
3      (
4        (or (= n 0)(null lst))
5        0
6      )
7      (
8        (+
9          (car lst)
10         (sum_to_n (cdr lst) (- n 1))
11       )
12     )
13   )
14 )
```

lst - входной список, **n** - индекс элемента, до которого нужно производить сложение.

Условием выхода из рекурсии является достижение конца списка(первый аргумент функции - nil) или достижение нужного элемента(второй аргумент функции - 0) - возвращается 0. Иначе осуществляется рекурсивный вызов текущей функции для хвоста первого аргумента и второго аргумента, уменьшенного на 1, и возвращается сумма результата рекурсивного вызова и головы первого аргумента.

Примеры работы:

```
> (sum_to_n nil 1)
0
> (sum_to_n '(1) 1)
1
> (sum_to_n '(1 2 3 4 5 6) 3)
6
```

Задание 13

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

Листинг 13: Функция-обертка для получения последнего нечетного числа

```
(defun get_last_odd (lst)
  (get_odd lst nil)
)
```

lst - входной список.

Листинг 14: Функция получения последнего нечетного числа

```
(defun get_odd (lst num)
  (cond
    (
      (null lst)
      num
    )
    (
      (oddp (car lst))
      (get_odd (cdr lst) (car lst))
    )
    (
      (get_odd (cdr lst) num)
    )
  )
)
```

lst - входной список, **num** - последнее найденное нечетное число.

Условием выхода из рекурсии является достижение конца списка(первый аргумент - nil) - возвращается второй аргумент. Иначе осуществляется проверка

на нечетность головы первого аргумента, если он нечетный, то осуществляется рекурсивный вызов текущей функции для хвоста первого аргумента и головы первого аргумента, если четный - осуществляется рекурсивный вызов для хвоста первого аргумента и второго аргумента.

Примеры работы:

```
> (get_last_odd nil)
NIL
> (get_last_odd '(1))
1
> (get_last_odd '(1 2 3))
3
> (get_last_odd '(1 2 3 4 5 6))
5
```

Задание 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.

Листинг 15: Функция получения квадратов чисел из списка

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

lst - входной список.

Условием выхода из рекурсии является достижение конца списка (аргумент - nil) - возвращается nil. Иначе осуществляется рекурсивный вызов текущей функции для хвоста аргумента, и возвращатеся списочная ячейка, указатель головы которой указывает на голову аргумента, умноженную на саму себя, а указатель хвоста - на результат рекурсивного вызова.

Примеры работы:

```
> (get_squares nil)
NIL
> (get_squares '(1))
(1)
> (get_squares '(1 2))
(1 4)
> (get_squares '(1 2 3 4 5))
(1 4 9 16 25)
```

Задание 15

Написать функцию с именем select-odd, которая из заданного списка выбирает все нечетные числа.

С использованием функционалов:

Листинг 16: Функция получения всех нечетных чисел из списка

```
(defun select-odd (lst)
  (mapcan #'(lambda (x)
      (cond
        (
          (and (numberp x)(oddp x))
          (cons x nil)
        )
        (
          (listp x)
          (select-odd x)
        )
      )
    ) lst
  )
)
```

lst - входной список.

С помощью mapcan осуществляется проверка типа каждого элемента:

- если он является нечетным числом, то возвращается списочная ячейка, указатель головы которой указывает на элемент, а хвоста - на nil;
- если он является списком, то для него осуществляется рекурсивный вызов текущей функции;
- во всех иных случаях возвращается nil.

Хвостовая рекурсия:

Листинг 17: Функция-обертка получения всех нечетных чисел из списка

```
(defun select-odd (lst)
  (reverse (select-odd-helper lst nil))
)
```

lst - входной список.

Листинг 18: Функция получения всех нечетных чисел из списка

```
(defun select-odd-helper (lst res)
  (cond
    (
      (null lst)
      res
    )
    (
      (listp (car lst))
      (select-odd-helper
        (cdr lst)
        (select-odd-helper (car lst) res))
    )
    (
      (and
        (numberp (car lst))
        (oddp (car lst))
      )
      (select-odd-helper (cdr lst) (cons (car lst) res))
    )
    (
      (select-odd-helper (cdr lst) res)
    )
  )
)
```

lst - входной список, **res** - результирующий список нечетных чисел.

Условием выхода из рекурсии является достижение конца списка(первый аргумент - nil) - возвращается второй аргумент. Если голова первого аргумента список, то осуществляется рекурсивный вызов текущей функции для хвоста первого аргумента и рекурсивного вызова, который осуществляется для головы первого аргумента и второго аргумента. Если голова первого аргумента - нечетное число, то осуществляется рекурсивный вызов текущей функции для хвоста списка и списка, указатель на голову которого указывает на голову первого аргумента, а указатель на хвост - на второй аргумент. Иначе осуществляется рекурсивный вызов текущей функции для хвоста списка и второго аргумента.

При данной реализации результирующий список оказывается перевернутым, поэтому к нему применяется функция reverse.

Примеры работы:

```
> (select-odd nil)
NIL
> (select-odd '(1))
(1)
> (select-odd '(1 2 3 4 5))
(1 3 5)
> (select-odd '((1 2 (3) 4) a (5 (7))))
(1 3 5 7)
```

Теоретические вопросы

Способы организации повторных вычислений в Lisp

Для организации многократных вычислений в Lisp могут быть использованы функционалы - функции, которые в качестве своего аргумента принимают функциональный объект — функцию, имеющую имя (глобально определенную функцию), или функцию, не имеющую имени (локально определенную функцию).

Также для организации многократных вычислений в Lisp может быть использована рекурсия. Рекурсия — это ссылка на определяемый объект во время его определения.

Что такое рекурсия?Классификация рекурсивных функций в Lisp

Рекурсия — это ссылка на определяемый объект во время его определения. В LISP существует классификация рекурсивных функций:

- простая рекурсия - один рекурсивный вызов в теле
- рекурсия первого порядка - рекурсивный вызов встречается несколько раз
- взаимная рекурсия - используется несколько функций, рекурсивно вызывающих друг друга.

Различные способы организации рекурсивных функций и порядок их реализации

Способы организации рекурсивных функций:

- Хвостовая рекурсия

  Результат формируется не на выходе из рекурсии, а на входе в рекурсию, все действия выполняются до ухода на следующий шаг рекурсии.

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

- Рекурсия по нескольким параметрам

```
(defun fun (n x)
(cond (end_test end_value)
( t (fun (reduced_n) (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. Количество условий выхода также может зависеть от задачи.

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

Способы повышения эффективности реализации рекурсии

Рекомендуется организовывать и отлаживать реализацию отдельных подзадач исходной задачи, обращая внимание на эффективность реализации и качество работы, а потом, при необходимости, встраивать эти функции в более крупные, возможно в виде лямбда-выражений.

Для повышения эффективности рекурсии необходимо правильно организовывать условия выхода из нее. Основное правило: при построении условного выражения первое условие - это всегда выход из рекурсии, но если условий выхода несколько, то надо думать о порядке их следования. Некачественный выход из рекурсии может привести к переполнению памяти из-за "лишних"рекурсивных

вызовов. Кроме того возможна потеря аргумента - кажется что функция возвращает результат и он используется, но на деле результат теряется и ответ неверен.

В целях повышения эффективности рекурсивных функций рекомендуется формировать результат не на выходе из рекурсии, а на входе в рекурсию, все действия выполняя до ухода на следующий шаг рекурсии.