Министерство образования и науки 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»

Отчет по лабораторной работе № 10

По курсу: "Функциональное и логическое программирование"

Студент:

Турсунов Жасурбек Рустамович

Группа: ИУ7-66Б

Преподователи:

Толпинская Наталья Борисовна

Строганов Юрий Владимирович

Москва, 2021 г.

Содержание

Введение **2**

Задание 1 **3**

Задание 2 **4**

Задание 3 **5**

Задание 4 **7**

Задание 5 **8**

Задание 6 **9**

Задание 7 **10**

Задание 8 **12**

Цель работы: приобрести навыки организации сложных рекурсивных функций с использованием функционалов.

Задачи работы: изучить способы применения функционалов и рекурсии при обработке одноуровневых и структурированных списков.

Введение

Для организации многократных вычислений в Lisp могут быть использованы функционалы - функции, которые особым образом обрабатывают свои аргументы. Функционалы - это функции более высокого порядка, так как они в качестве своего первого аргумента принимают функциональный объект - функцию, имеющую имя(глобально определенную функцию), или функцию, не имеющею имени(локально определенную функцию). При использовании рекурсии - необходимо обеспечить эффективность работы, путем использования хвостовой рекурсии. В рекурсивных функциях могут быть использованы дополнительные функции - не в аргументах вызова, а вне них. Рекурсивные вызовы могут быть организованы как в одной ветке, так и в нескольких ветках программы. Рекурсивные функции могут вызывать друг друга. функционалы, являющиеся предикатами, функционалы, использующие предикаты в качестве функционального объекта

Задание 1

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

Хвостовая рекурсия:

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

Листинг 1: Функция-обертка для вычисления суммы чисел списка

lst - входной список.

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

Листинг 2: Функция вычисления суммы чисел списка

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:

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

Задание 2

Написать рекурсивную версию с именем rec-nth функции nth.

```
(defun rec_nth (lst n)
    (cond
        (
            (null lst)
            nil
        )
        (
            (= n 0)
            (car lst)
        )
        (
            (rec_nth (cdr lst) (- n 1))
```

```
        )
    )
)
```

Листинг 3: Рекурсивная версия функции nth

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:

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

Задание 3

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

Хвостовая рекурсия:

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

Листинг 4: Функция-обертка проверки на нечетность всех элементов списка

lst - входной список.

```
(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))
                (oddp (car lst))
            )
            (alloddr-helper (cdr lst) isodd)
        )
    )
)
```

Листинг 5: Функция проверки на нечетность всех элементов списка

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 ар-

гумента 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:

```
> (alloddr nil)
T
> (alloddr '(1))
T
> (alloddr '(1 2))
NIL
> (alloddr '(1 3))
T
> (alloddr '(1 (3 (5) 7) 9))
T
> (alloddr '(1 (3 nil) 2))
NIL
```

Задание 4

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

```
(defun get_last (lst)
    (cond
        (
            (null (cdr lst))
            (car lst)
        )
        (
            (get_last (cdr lst))
        )
    )
)
```

Листинг 6: Функция получения последнего элемента списка

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:

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

Задание 5

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

```
(defun sum_to_n (lst n)
    (cond
        (
            (or (= n 0)(null lst))
            0
        )
        (
            (+
                (car lst)
                (sum_to_n (cdr lst) (- n 1))
            )
        )
    )
```

```
)
```

Листинг 7: Функция вычисления суммы элементов от 0 до n-го

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 6

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

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

Листинг 8: Функция-обертка для получения последнего нечетного числа

lst - входной список.

```
(defun get_odd (lst num)
    (cond
        (
            (null lst)
            num
```

```
        )
        (
            (oddp (car lst))
            (get_odd (cdr lst) (car lst))
        )
        (
            (get_odd (cdr lst) num)
        )
    )
)
```

Листинг 9: Функция получения последнего нечетного числа

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 7

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

Листинг 10: Функция получения квадратов чисел из списка

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 8

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

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

Листинг 11: Функция получения всех нечетных чисел из списка

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.

Примеры работы:

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

```
6     (1 3 5)
7     > (select-odd '((1 2 (3) 4) a (5 (7))))
8     (1 3 5 7)
```