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»

Отчет по лабораторной работе №10 по курсу "Функциональное и логическое программирование"

Тема Вложенные рекурсия и функционалы

Студент Пересторонин П.Г.

Группа ИУ7-63Б

Оценка _____

Преподаватель Толпинская Н. Б.

Москва — 2021 г.

Оглавление

1 Задания

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

```
(defun rec-add-internal (lst acc)
  (cond ((null lst) acc)
        (T (rec-add-internal (cdr lst) (+ acc (car lst))))))
(defun rec-add (lst)
  (rec-add-internal lst 0))
```

1.2 Написать рекурсивную функцию с именем rec-nth функции nth

```
(defun rec-nth (n lst)
  (and lst (cond ((zerop n) (car lst))
                 (T (rec-nth (- n 1) (cdr lst))))))
```

1.3 Написать рекурсивную функцию alloddr, которая возвращает t, когда все элементы списка нечётные

```
(defun alloddr (lst)
  (or (null lst)
      (and (oddp (car lst))
           (alloddr (cdr lst)))))
```

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

```
(defun last-rec (lst)
  (cond ((null (cdr lst)) (car lst))
        (T (last-rec (cdr lst)))))
```

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

1.5.1 От n-аргумента функции, до последнего $>= 0$

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

1.5.2 От n-аргумента функции до m-аргумента с шагом d

```
(defun sum-nmd (n m d lst)
  (cond ((> n m) 0)
        (T (+ (nth n lst) (sum-nmd n (- m d) d (nthcdr d lst))))))
```

1.6 Написать рекурсию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции

```
(defun last-odd-internal (lst cur-odd)
  (cond ((null lst) cur-odd)
        (T (cond ((oddp (car lst)) (last-odd-internal (cdr lst) (car lst)))
                 (T (last-odd-internal (cdr lst) cur-odd))))))
(defun last-odd (lst)
  (last-odd-internal lst nil))
```

1.7 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 cons-square (lst)
  (and lst (cons ((lambda (x) (* x x)) (car lst))
                 (cons-square (cdr lst)))))
```

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

Варианты:

1. select-even;

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

```
(defun select-odd (lst)
  (mapcan #'(lambda (x) (if (oddp x) (list x))) lst))
(defun select-even (lst)
  (mapcan #'(lambda (x) (if (evenp x) (list x))) lst))

(defun sum-all-odd (lst)
  (apply #'+ (select-odd lst)))
(defun sum-all-even (lst)
  (apply #'+ (select-even lst)))
```

1.9 Создать и обработать смешанный структурированный список с информацией:

- ФИО;
- зарплата;
- возраст;
- категория (квалификация).

Изменить зарплату в зависимости от заданного условия, и подсчитать суммарную зарплату. Использовать композиции функций.

Исходные данные:

```
(setf people (list
              (list (cons 'fio "Ivanov Ivan Ivanovich")
               (cons 'salary 1000)
               (cons 'age 18)
               (cons 'category "programmer"))
              (list (cons 'fio "Petrov Ivan Ivanovich")
               (cons 'salary 2000)
               (cons 'age 28)
               (cons 'category "builder"))
              (list (cons 'fio "Petrov Petr Petrovich")
               (cons 'salary 3000)
               (cons 'age 32)
               (cons 'category "football manager"))
              ))
```

Изменение зарплаты в зависимости от заданного условия:

```
(defun salary-wrapper (salary-func man)
  (let ((salary-item (assoc 'salary man)))
    (setf (cdr salary-item) (funcall salary-func (cdr salary-item)))))

(defun change-salaries (cond-func change-func lst)
  (mapcar #'(lambda (man)
              (cond ((funcall cond-func man) (salary-wrapper change-func man))
                    (T man)))
          lst))
```

Подсчет суммарной зарплаты:

```
(defun count-salaries (people)
  (reduce #'(lambda (acc man)
              (+ acc (cdr (assoc 'salary man))))
          (cons 0 people)))
```

Пример вызова:

```
(change-salaries #'(lambda (man) (equal (cdr (assoc 'category man)) "programmer"))
                 #'(lambda (salary) (* salary 99))
                 people)
(count-salaries people)
```