



КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Тема Вложенные рекурсия и функционалы

Студент Пересторонин П.Г.

Группа ИУ7-63Б

Оценка

Преподаватель Толпинская Н. Б.

Москва — 2021 г.

Оглавление

1	Задания	2
1.1	Написать рекурсивную версию (с именем rec-add) вычисления суммы чисел заданного списка	2
1.2	Написать рекурсивную функцию с именем rec-nth функции nth	2
1.3	Написать рекурсивную функцию alloddr , которая возвращает t , когда все элементы списка нечётные	2
1.4	Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка-аргумента	3
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	3
1.5.1	От n -аргумента функции, до последнего ≥ 0	3
1.5.2	От n -аргумента функции до m -аргумента с шагом d	3
1.6	Написать рекурсию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции	4
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	4
1.8	Написать функцию с именем select-odd , которая из заданного списка выбирает все нечетные числа	4
1.9	Создать и обработать смешанный структурированный список с информацией:	5

1 Задания

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

```
1 (defun rec-add-internal (lst acc)
2   (cond ((null lst) acc)
3         (T (rec-add-internal (cdr lst) (+ acc (car lst))))))
4 (defun rec-add (lst)
5   (rec-add-internal lst 0))
```

1.2 Написать рекурсивную функцию с именем `rec-nth` функции `nth`

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

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

```
1 (defun alloddr (lst)
2   (or (null lst)
3       (and (oddp (car lst))
4            (alloddr (cdr lst)))))
```

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

```
1 (defun last-rec (lst)
2   (cond ((or (null lst) (null (cdr lst))) (car lst)))
3         (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

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

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

```
1 (defun sum-nmd-internal (m d lst)
2   (cond ((or (null lst) (< m 0)) 0)
3         (T (+ (car lst) (sum-nmd-internal (- m d) d (nthcdr d lst)))))
4 (defun sum-nmd (n m d lst)
5   (sum-nmd-internal (- (- m 1) d) d (nthcdr n lst)))
```

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

```
1 (defun last-odd-internal (lst cur-odd)
2   (cond ((null lst) cur-odd)
3         (T (cond ((oddp (car lst)) (last-odd-internal (cdr lst) (car lst)))
4                  (T (last-odd-internal (cdr lst) cur-odd)))))
5 (defun last-odd (lst)
6   (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

```
1 (defun cons-square (lst)
2   (and lst (cons ((lambda (x) (* x x)) (car lst))
3                  (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.

```
1 (defun select-odd (lst)
2   (mapcan #'(lambda (x) (if (oddp x) (list x))) lst))
3 (defun select-even (lst)
4   (mapcan #'(lambda (x) (if (evenp x) (list x))) lst))
5
6 (defun sum-all-odd (lst)
7   (apply #'+ (select-odd lst)))
8 (defun sum-all-even (lst)
9   (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:

```
1 (setf people (list
2   (list (cons 'fio "Ivanov_Ivan_Ivanovich")
3     (cons 'salary 1000)
4     (cons 'age 18)
5     (cons 'category "programmer")))
6   (list (cons 'fio "Petrov_Ivan_Ivanovich")
7     (cons 'salary 2000)
8     (cons 'age 28)
9     (cons 'category "builder")))
10  (list (cons 'fio "Petrov_Petr_Petrovich")
11    (cons 'salary 3000)
12    (cons 'age 32)
13    (cons 'category "football_manager")))
14  ))
```

Изменение зарплаты в зависимости от заданного условия:

```
1 (defun salary-wrapper (salary-func man)
2   (let ((salary-item (assoc 'salary man)))
3     (setf (cdr salary-item) (funcall salary-func (cdr salary-item)))))
4
5 (defun change-salaries (cond-func change-func lst)
6   (mapcar #'(lambda (man)
7               (cond ((funcall cond-func man) (salary-wrapper change-func man))
8                     (T man)))
9     lst))
```

Подсчет суммарной зарплаты:

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

Пример вызова:

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