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.

Группа ИУ7-63Б

Студент Сукочева А.

Преподаватель Толпинская Н.Б.

Преподаватель Строганов Ю. В.

Москва — 2021 г.

Практическая часть

Задание 1. Написать рекурсивную версию (с именем rec-add) вычисления суммы чисел заданного списка.

```
(defun add-rec (lst sum)
    (cond ((null lst) sum)
    (T (add-rec (cdr lst) (+ sum (car lst)) )) ))

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

Пример использования:

```
(rec-add '(1 2 3)) ;; 6
```

Для структурированного списка:

```
(defun add-rec (lst sum)
    (cond ((null lst) sum)
    ((listp (car lst)) (+ (add-rec (car lst) 0) (add-rec (cdr lst) sum)))
    (T (add-rec (cdr lst) (+ (car lst) sum)))))

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

Пример использования:

```
(rec-add '(1 2 (3 4 5) 6)) ;; => 21
```

Задание 2. Написать рекурсивную версию с именем `rec-nth` функции `nth`.

```
(defun rec-nth (n lst)
        (cond ((null lst) nil)
        ((= n 0) (car lst))
        (T (rec-nth (- n 1) (cdr lst))) ))
```

Пример использования:

```
(nth 0 '(a w c))         ;; => A
(rec-nth 0 '(a w c))     ;; => A

(nth 1 '(a b c))         ;; => B
(rec-nth 1 '(a b c))     ;; => B
```

Задание 3. 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)))))
```

Пример использования:

```
(alloddr '(2 1 3 5))    ;; Nil
(alloddr '(11 1 3 5))   ;; T
```

Для смешанного структурированного списка:

```
(defun alloddr (lst)
    (cond ((null lst) T)
    ((symbolp (car lst)) (alloddr (cdr lst)))
    ((listp (car lst)) (and (alloddr (car lst)) (alloddr (cdr lst))))
    ((evenp (car lst)) nil)
    (T (alloddr (cdr lst)))))
```

Пример использования:

```
(alloddr '(1 1 (3 5 7 (9))))    ;; T
(alloddr '(1 1 a (3 5 7 (8))))  ;; Nil
```

Задание 4. Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка — аргументы.

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

Пример использования:

```
(my-last '(1 2 3 4 5 6))     ;; => 6
(my-last ())                 ;; => NIL
(my-last '(a b c d (e f)))   ;; => (E F)
```

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.

Вариант дополняемой рекурсии:

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

Пример использования:

```
(sum-rec '(1 2 3 4 5 6 7 8 9 10) 3) ;; => 6
```

Вариант хвостовой рекурсии:

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

(defun sum-rec (lst n)
        (f-sum-rec lst n 0))
```

Пример использования:

```
(sum-rec '(1 2 3 4 5 6 7 8 9 10) 3) ;; => 6
```

Вариант от n-аргумента функции до последнего $>= 0$.

```
(defun sum-n-last (lst n)
    (cond ((= 0 n) (sum-rec lst (length lst)))
    (T (sum-n-last (cdr lst) (- n 1))) ))
```

Пример использования:

```
(sum-n-last '(1 2 3 4 5 6 7 8 9 10) 5) ;; => 40
```

Вариант от n-аргумента функции до m с шагом d

```
(defun do-step (lst step)
    (cond ((= 0 step) lst)
    (T (do-step (cdr lst) (- step 1)))))
```

```
(do-step '(1 2 3 4 5 6) 2) ;; => (3 4 5 6)
```

```
(defun f-sum-rec-step (lst n step sum)
    (cond ((or (= 0 n) (> step n)) sum)
    (T (f-sum-rec-step (do-step lst step) (- n step) step (+ sum (car lst))))
       ))

(defun sum-n-m-d (lst n m d)
    (cond ((= 0 n) (f-sum-rec-step lst m d 0))
    (T (sum-n-m-d (cdr lst) (- n 1) m d)) ))
```

Пример использования (Нумерация с нуля!):

```
(sum-n-m-d '(1 2 3 4 5 6 7 8 9 10) 0 10 1) ;; => 55
(sum-n-m-d '(1 2 3 4 5 6 7 8 9 10) 1 6 3)  ;; => 7
```

Задание 6. Написать рекурсивную функцию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции.

```
(defun last-odd-rec (lst last-odd-number)
    (cond ((null lst) last-odd-number)
    (T (last-odd-rec (cdr lst)
        (if (oddp (car lst)) (car lst) last-odd-number))) ))

(defun last-odd (lst)
    (last-odd-rec lst Nil))
```

Пример использования:

```
(last-odd '(1 2 3 4 5 6 7))      ;; => 7
(last-odd '(1 2 3 4 5 6))        ;; => 5
(last-odd '(2 4 6 8))            ;; => NIL
(last-odd '(1 3 5))              ;; => 5
(last-odd '(2 4 3 6 8 10))       ;; => 3
```

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 square-lst (lst)
    (cond ((null lst) Nil)
    (T (cons (* (car lst) (car lst)) (square-lst (cdr lst)))) ))
```

Пример использования:

```
(square-lst '(1 2 3 4 5 6)) ;; => (1 4 9 16 25 36)
```

Задание 8. Написать функцию с именем `select-odd`, которая из заданного списка выбирает все нечетные числа.

Вариант 1.a: `select-odd`

```
(defun select-odd (lst)
    (cond ((null lst) Nil)
    (T (if
        (oddp (car lst)) (cons (car lst) (select-odd (cdr lst)))
        (select-odd (cdr lst)))) ))
```

```
(select-odd '(1 2 3 4 5 6 7 8 9 1)) ;; => (1 3 5 7 9 1)
```

Вариант 1.b: `select-even`

```
(defun select-even (lst)
    (cond ((null lst) Nil)
    (T (if (evenp (car lst))
        (cons (car lst) (select-even (cdr lst)))
        (select-even (cdr lst)))) ))
```

Пример использования:

```
(SELECT-EVEN '(1 2 3 4 5 6 7 8 9 1)) ;; => (2 4 6 8)
```

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

Сумма нечетных чисел.

```
(defun sum-all-odd-rec (lst sum)
        (cond ((null lst) sum)
        (T (sum-all-odd-rec (cdr lst)
                (if (oddp (car lst)) (+ (car lst) sum) sum))) ))

(defun sum-all-odd (lst)
        (sum-all-odd-rec lst 0))
```

Пример использования:

```
(sum-all-odd '(1 2 3 4 5)) ;; => 9
```

Для смешанного стуктурированного списка:

```
(defun sum-all-odd-rec (lst sum)
    (cond ((null lst) sum)
    ((symbolp (car lst)) (sum-all-odd-rec (cdr lst) sum))
    ((listp (car lst))
    (+ (sum-all-odd-rec (car lst) 0) (sum-all-odd-rec (cdr lst) sum)))
    (T (sum-all-odd-rec (cdr lst)
        (if (oddp (car lst)) (+ (car lst) sum) sum))) ))

(defun sum-all-odd (lst)
    (sum-all-odd-rec lst 0))
```

Пример использования:

```
(sum-all-odd '((1 2) (3) abc a (b c 4 5) T a)) ;; => 9
```

Задание 9. Обработка списка с информацией. Создать и обработать смешанный структурированный список с информацией: ФИО, зарплата, возраст, категория(квалификация).

(ФИО зарплата возраст категория)

```
(defun get-fio (lst) (car lst))
(defun get-salary (lst) (cadr lst))
(defun get-age (lst) (caddr lst))
(defun get-category (lst) (car lst))
```

Изменить зарплату, в зависимости от заданного условия.

Вспомогательная функция, для изменения зарплаты у сотрудника.

```
(defun set-new-salary (man new-salary)
    (list (get-fio man) new-salary (get-age man) (get-category man)) )

(defun change-salary (condition lst new-salary)
    (if (funcall condition lst) (set-new-salary lst new-salary) lst))
```

Пример использования:

```
(change-salary (lambda (man) (< (get-salary man) 135000 ))
    '(Alice 14 20 1) 14000)  ;; Result: (ALICE 14000 20 ALICE)
```

```
(defun change-all-salary (workers condition new-salary)
    (cond ((null workers) Nil)
    (T (cons (change-salary condition (car workers) new-salary)
    (change-all-salary (cdr workers) condition new-salary))) ))
```

Пример использования: Меняем у зарплату у всех, у кого она меньше, чем 13500 на 50000

```
(change-all-salary
        '((Alice 12000 20 1)
        (Pasha 12000 20 2)
        (Nastya 80000 21 3))
        (lambda (man) (< (get-salary man) 13500 ))
        50000)
;; Result: ((ALICE 50000 20 ALICE) (PASHA 50000 20 PASHA) (NASTYA 80000 21 3))
```

Подсчитать сумарную зарплату:

```
(defun sum-salary-rec (workers sum)
    (cond ((null workers) sum)
    (T (sum-salary-rec (cdr workers) (+ sum (get-salary(car workers))))) ))

(defun sum-salary (workers)
    (sum-salary-rec workers 0))
```

Пример использования:

```
(sum-salary '((Alice 120 20 1)
    (Pasha 120 20 2)
    (Nastya 150 21 3))) ;; => 390
```

Использовать композицию функций:

```
(sum-salary (change-all-salary
        '((Alice 12000 20 1)
        (Pasha 12000 20 2)
        (Nastya 80000 21 3))
        (lambda (man) (< (get-salary man) 13500 ))
        50000)) ;; => 180000
```