|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

*к лабораторной работе №10*

*По курсу: «Функциональное и логическое программирование»*

**Тема: «**Вложенные рекурсия и функционалы».

Студент: Якуба Д.В.

Группа: ИУ7-63Б

Преподаватели: Толпинская Н. Б.,

Строганов Ю. В.

Москва, 2021 г.

# Практическая часть

Задание 1. Написать рекурсивную версию (с именем rec-add) вычисления суммы чисел заданного списка. Например: (rec-add (2 4 6)) -> 12.

Решение:

|  |
| --- |
| (*defun* rec-add-inner (lst acc)      (*cond*          ((null lst) acc)          ((listp (car lst)) (rec-add-inner (cdr lst) (+ acc (rec-add-inner (car lst) 0))))          ((numberp (car lst)) (rec-add-inner (cdr lst) (+ acc (car lst))))          (t (rec-add-inner (cdr lst) acc))))  (*defun* rec-add (lst)      (rec-add-inner lst 0)) |

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

(rec-add '(1 2 (7 8) 3)) -> 21;

(rec-add '(1 2 (7 oh 8) i 3 ((4 just wanna ((sleep)))))) -> 25;

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

Решение:

|  |
| --- |
| (*defun* rec-nth (index lst)      (*cond*          ((null lst) nil)          ((zerop index) (car lst))          (t (rec-nth (- index 1) (cdr lst))))) |

(rec-nth 3 '(0 1 2 3)) -> 3;

(rec-nth 3 '(0 1 2 (3 3))) -> (3 3);

(rec-nth 3 '((3 3))) -> nil;

(rec-nth -3 '((3 3))) -> nil;

Задание 3. Написать рекурсивную функцию alloddr, которая возвращает t, когда все элементы списка нечетные.

Решение:

|  |
| --- |
| ; без работы с структурированными смешанными списками  (*defun* allodr-tail-inner (lst cur-bool)      (*cond*          ((null cur-bool) nil)          ((null lst))          (t (allodr-tail-inner (cdr lst) (oddp (car lst))))))  (*defun* allodr-tail (lst)      (allodr-tail-inner lst t))  ; для работы с структурированными смешанными списками  (*defun* allodr-tail-inner (lst cur-bool)      (*cond*          ((null cur-bool) nil)          ((null lst))          ((listp (car lst)) (*and* (allodr-tail-inner (car lst) t) (allodr-tail-inner (cdr lst) cur-bool)))          ((numberp (car lst)) (allodr-tail-inner (cdr lst) (oddp (car lst))))          (t (allodr-tail-inner (cdr lst) cur-bool))))  (*defun* allodr-tail (lst)      (allodr-tail-inner lst t)) |

(allodr-tail-inner '(1 2 (3 4 (5 6)))) -> NIL;

(allodr-tail-inner '(1 1 (3 3 (5 5)))) -> T;

(allodr-tail-inner '(1 1 (3 (7) ((((((9)))))) 3 (5 5)))) -> T;

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

Решение:

|  |
| --- |
| (*defun* my-last (lst)      (*cond*          ((null (cdr lst)) lst)          (t (my-last (cdr lst))))) |

(my-last '(1 2 3 4)) -> (4);

(my-last '(1 2 3 (2 3))) -> ((2 3));

(my-last '(1 2 3 ())) -> (NIL);

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:

1) от n-аргумента функции до последнего ,

2) от n-аргумента функции до m-аргумента с шагом d.

Решение:

|  |
| --- |
| ; от 0 до n-го)  (*defun* sum-to-n-inner (lst to-n acc)      (*cond*          ((*or* (null lst) (< to-n 0)) acc)          (t (sum-to-n-inner (cdr lst) (- to-n 1) (+ acc (car lst))))))  (*defun* sum-to-n (lst to-n)      (sum-to-n-inner lst to-n 0))  ; от n-го до первого < 0  (*defun* sum-to-lz-inner (lst acc)      (*cond*          ((*or* (null lst) (< (car lst) 0)) acc)          (t (sum-to-lz-inner (cdr lst) (+ acc (car lst))))))  (*defun* sum-to-lz (lst)      (sum-to-lz-inner lst 0))  ; от n до первого < 0  (*defun* sum-from-n-to-lz-inner (lst from-n acc)      (*cond*          ((null lst) acc)          ((> from-n 0) (sum-from-n-to-lz-inner (cdr lst) (- from-n 1) 0))          ((< (car lst) 0) acc)          (t (sum-from-n-to-lz-inner (cdr lst) 0 (+ acc (car lst))))))  (*defun* sum-from-n-to-lz (lst from-n)      (sum-from-n-to-lz-inner lst from-n 0))  ; от n-го до m с шагом h  ; Функция выполнения шага  (*defun* make-step (lst step-h)      (*cond*          ((> step-h 0) (make-step (cdr lst) (- step-h 1)))          (t lst)))  ; Суммируем и делаем шаг  (*defun* sum-to-m-with-step (lst to-m step-h acc)      (*cond*          ((*or* (null lst) (< to-m 0)) acc)          (t (sum-to-m-with-step (make-step lst step-h) (- to-m step-h) step-h (+ acc (car lst))))))  ; Проход до n-го элемента списка и вызов подсчитывающей функции  (*defun* sum-from-to-step-inner (lst from-n to-m step-h)      (*cond*          ((null lst) 0)          ((zerop from-n) (sum-to-m-with-step lst to-m step-h 0))          (t (sum-from-to-step-inner (cdr lst) (- from-n 1) to-m step-h))))  ; Подготовка вызова  (*defun* sum-from-to-step (lst from-n to-m step-h)      (*cond*          ((*or* (null lst) (= step-h 0) (< from-n 0) (< to-m 0)) 0)          ((< from-n to-m) (sum-from-to-step-inner lst from-n (- to-m from-n) step-h))          (t (sum-from-to-step-inner lst to-m (- from-n to-m) (abs step-h))))) |

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

Решение:

|  |
| --- |
| (*defun* ret-last-odd-inner (lst num)      (*cond*          ((null lst) num)          (t (ret-last-odd-inner (cdr lst) (*cond*                                              ((oddp (car lst)) (car lst))                                              (t num))))))  (*defun* ret-last-odd (lst)      (ret-last-odd-inner lst nil)) |

(ret-last-odd '(1 2 3 4 5)) -> 5;

(ret-last-odd '(1 2 3 4 2)) -> 3;

(ret-last-odd '(2 2 2 4 2)) -> nil;

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.

Решение:

|  |
| --- |
| (*defun* get-sqr-list (lst)      (*cond*          ((null lst) nil)          (t (cons (\* (car lst) (car lst)) (get-sqr-list (cdr lst)))))) |

(get-sqr-list '(1 2 3 4)) -> (1 4 9 16);

(get-sqr-list '(1 -2 -3 4)) -> (1 4 9 16);

(get-sqr-list '(8 3 7 2)) -> (64 9 49 4);

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

(Вариант 1: select-even,

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

|  |
| --- |
| (*defun* select-odd (lst)      (*cond*          ((null lst) nil)          ((oddp (car lst)) (cons (car lst) (select-odd (cdr lst))))          (t (select-odd (cdr lst)))))  (*defun* sum-all-odd-inner (lst acc)      (*cond*          ((null lst) acc)          ((*and* (numberp (car lst)) (oddp (car lst))) (sum-all-odd-inner (cdr lst) (+ acc (car lst))))          (t (sum-all-odd-inner (cdr lst) acc))))  (*defun* sum-all-odd (lst)      (sum-all-odd-inner lst 0))  (*defun* select-even (lst)      (*cond*          ((null lst) nil)          ((evenp (car lst)) (cons (car lst) (select-even (cdr lst))))          (t (select-even (cdr lst)))))  (*defun* sum-all-even-inner (lst acc)      (*cond*          ((null lst) acc)          ((*and* (numberp (car lst)) (evenp (car lst))) (sum-all-even-inner (cdr lst) (+ acc (car lst))))          (t (sum-all-even-inner (cdr lst) acc))))  (*defun* sum-all-even (lst)      (sum-all-even-inner lst 0)) |

(select-even '(1 2 3 4 5 6)) -> (2 4 6);

(select-even '(-3 -2 -1 0 1 2 3 4 5 6)) -> (-2 0 2 4 6);

(select-odd '(1 2 3 4 5 6)) -> (1 3 5);

(select-odd '(-3 -2 -1 0 1 2 3 4 5 6)) -> (-3 -1 1 3 5);

(sum-all-odd '(1 2 3 4 5 6)) -> 9;

(sum-all-odd '(-3 -2 -1 0 1 2 3 4 5 6)) -> 5;

(sum-all-even '(1 2 3 4 5 6)) -> 12;

(sum-all-even '(-3 -2 -1 0 1 2 3 4 5 6)) -> 10;

Задание 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.

Решение:

|  |
| --- |
| '(      ((fio . Jasurbek\_Mihailov) (salary . 3000) (age . 30) (pos . Manager))      ((fio . Torwalds\_Linus) (salary . 3) (age . 25) (pos . Director))      ((fio . Karl\_Marx) (salary . 300) (age . 55) (pos . Manager))      ((fio . Dmitry\_Yakuba) (salary . 500) (age . 20) (pos . Suicider)))  (*defun* get-val-by-key (key row)      (*cond*          ((null row) nil)          ((equal (caar row) key) (cdar row))          (t (get-val-by-key key (cdr row)))))  (*defun* by-key-no-cdr (key row)      (*cond*          ((null row) nil)          ((equal (caar row) key) (car row))          (t (by-key-no-cdr key (cdr row)))))  (*defun* change-salary-with-cond (table col-name predicate salaryFun)      (*cond*          ((null table) 'done)          ((funcall predicate (get-val-by-key col-name (car table)))              (*and*                  (rplacd (by-key-no-cdr 'salary (car table)) (funcall salaryFun (get-val-by-key 'salary (car table))))                  (change-salary-with-cond (cdr table) col-name predicate salaryFun)))          (t (change-salary-with-cond (cdr table) col-name predicate salaryFun))))  (change-salary-with-cond table 'pos #'(*lambda* (got-pos) (equal 'Manager got-pos)) #'(*lambda* (sal) (*setf* sal (/ sal 2))))  (*defun* get-sum-salary-inner (table acc)      (*cond*          ((null table) acc)          (t (get-sum-salary-inner (cdr table) (+ acc (get-val-by-key 'salary (car table)))))))  (*defun* get-sum-salary (table)      (get-sum-salary-inner table 0)) |

(change-salary-with-cond table 'pos #'(*lambda* (got-pos) (equal 'Manager got-pos)) #'(*lambda* (sal) (*setf* sal (/ sal 2)))) -> Урезание зарплаты в два раза для JASURBEK\_MIGAILOV и KARL\_MARX.

# Теоретическая часть

1. Классификация рекурсивных функций.

1) Простая рекурсия. Вызов является единственным.

2) Рекурсия второго порядка. Присутствует несколько рекурсивных вызовов.

3) Взаимная рекурсия. Несколько рекурсивных функций, которые могут друг друга вызывать.

4) Хвостовая рекурсия. При очередном рекурсивном вызове функции все действия до входа выполнены, а при выходе ничего более делать не потребуется.

5) Дополняемая рекурсия. Используется для обработки car и cdr указателей. Результат рекурсии используется в качестве аргумента другой функции:

|  |
| --- |
| (defun func(x)  (cond (end\_test end-value)  (t (add\_function add\_value (func changed\_x)))) |

Частные случаи: cons-дополняемая рекурсия, дополняемая функция встречается после прерывания рекурсии.