|  |  |
| --- | --- |
| 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 №9*

*По курсу: «Функциональное и логическое программирование»*

**Тема: «**Использование функционалов и рекурсии».

Студент: Якуба Д.В.

Группа: ИУ7-63Б

Преподаватели: Толпинская Н. Б.,

Строганов Ю. В.

Москва, 2021 г.

# Практическая часть

Задание 1. Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10. (Вариант: между двумя заданными границами)

Решение:

|  |
| --- |
| (*defun* select-rec (cur-lst down-limit up-limit)      (*cond*   ((null cur-lst) nil)              ((listp (car cur-lst)) (nconc (select-rec (car cur-lst) down-limit up-limit) (select-rec (cdr cur-lst) down-limit up-limit)))              ((*and* (numberp (car cur-lst))                    (<= (car cur-lst) up-limit)                    (>= (car cur-lst) down-limit))                    (cons (car cur-lst) (select-rec (cdr cur-lst) down-limit up-limit)))                    (t (select-rec (cdr cur-lst) down-limit up-limit))))  ; функция предподготовки  (*defun* select-between (lst fNum sNum)      (let ((down-limit (*cond* ((< fNum sNum) fNum) (t sNum)))            (up-limit (*cond* ((> fNum sNum) fNum) (t sNum))))              (select-rec lst down-limit up-limit))) |

(select-between '(1 2 (3 4 #'+ 3) ad 3 2 zxcv) 1 3) -> (1 2 3 3 3 2);

(select-between '(1 2 (3 4 #'+ 3) ad 3 2 zxcv) 3 1) -> (1 2 3 3 3 2);

(select-between '(1 2 (3 4 #'+ 3) ad 3 2 zxcv) -3 0) -> nil;

Задание 2. Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов. (Напомнит, что – это множество всевозможных пар (a, b), где a принадлежит А, b принадлежит B)

Решение:

|  |
| --- |
| (*defun* dec-Prod-iter (f-el s-lst)      (*cond* ((null s-lst) nil)            (t (cons (cons f-el (car s-lst)) (dec-Prod-iter f-el (cdr s-lst))))))  (*defun* dec-Prod (f-lst s-lst)      (*cond* ((null f-lst) nil)            (t (nconc (dec-Prod-iter (car f-lst) s-lst) (dec-Prod (cdr f-lst) s-lst)))))  (*defun* dec-Prod (f-lst s-lst)      (mapcan (*lambda* (f-el) (mapcar (*lambda* (s-el) (cons f-el s-el)) s-lst)) f-lst)) |

(dec-prod '(1 2 3 4) '(5 6 7)) -> ((1 . 5) (1 . 6) (1 . 7) (2 . 5) (2 . 6) (2 . 7) (3 . 5) (3 . 6) (3 . 7) (4 . 5) (4 . 6) (4 . 7));

(dec-prod '(1 2 (2 3) 2) '(5 6)) -> ((1 . 5) (1 . 6) (2 . 5) (2 . 6) ((2 3) . 5) ((2 3) . 6) (2 . 5) (2 . 6));

(dec-prod '(kill save) '(me us our\_souls)) -> ((KILL . ME) (KILL . US) (KILL . OUR\_SOULS) (SAVE . ME) (SAVE . US) (SAVE . OUR\_SOULS))

Задание 3. Почему так реализован reduce, в чем причина?

(reduce #’+ ()) -> 0;

Ответ: подобное поведение связано с тем, что «+» является специальной функцией, которая при количестве аргументов = 0 вернёт 0. При передаче reduce функций «-»и «/» будет возникать ошибка «invalid number of arguments».

Задание 4. Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list, т.е., например, для аргумента ((1 2) (3 4)) -> 4.

Решение:

|  |
| --- |
| ; В случае, если список состоит только из одноуровневых списков, можно не использовать рекурсию,  ; указанная функция работает для всех уровней  (*defun* sum-len (lst)      (reduce (*lambda* (accum cur-el) (*cond* ((listp cur-el) (+ accum (sum-len cur-el)))                                           (t (+ accum 1)))) (cons 0 lst)))  ; личная функция len, породирующая поведение length  (*defun* len (cdr-lst)      (*cond* ((null cdr-lst) 0)            (t (+ 1 (len (cdr cdr-lst))))))  ; С наложенными ограничениями (с использованием функционала):  (*defun* sum-len (lst)      (reduce          (*lambda* (accum-outer cur-el-outer)                  (+ accum-outer (len cur-el-outer)))          (cons 0 lst)))  ; С наложенными ограничениями (рекурсивно):  (*defun* sum-len (lst)      (*cond* ((null lst) 0)            ((numberp lst) 1)            (t (+ (sum-len (car lst)) (sum-len (cdr lst))))))  ; Ещё один подобный вариант с использованием собственной функции length  (*defun* sum-len (lst)      (*cond* ((null lst) 0)            (t (+ (len (car lst)) (sum-len (cdr lst)))))) |

Задание 5. Используя рекурсию, написать функцию, которая по исходному списку строит список квадратов чисел смешанного структурированного списка.

Решение:

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

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

(get-sqr-list '(1 2 3 (6 7) 4)) -> (1 4 9 36 49 16);

(get-sqr-list '(1 (oh) 2 (can i (get 9 some (sleep))) 3 (6 7) 4)) -> (1 4 81 9 36 49 16);

# 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.