|  |  |
| --- | --- |
| 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»

**Лабораторная работа № 9**

|  |  |
| --- | --- |
| **По дисциплине**: Функциональное и логическое программирование  **Тема: «**Использование управляющих структур, работа со списками**»**  **Студент** Унтилова А.О.  **Группа** ИУ7-66  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Толпинская Н.Б., Строганов Ю.В. |  |

Москва.

2021 г.

1. Написать функцию, которая выбирает из заданного списка только те числа,

которые больше 1 и меньше 10.

Рекурсия:

|  |
| --- |
| (defun in\_border (x a b)  (and (numberp x) (> x a) (< x b))  )  (defun get\_in\_border (lst a b)  (cond ((null lst) lst)  ((in\_border (car lst) a b) (cons (car lst) (get\_in\_border (cdr lst) a b)))  (t (get\_in\_border (cdr lst) a b))  )  ) |

Функционал:

|  |
| --- |
| (defun is\_between (left right a)  (if (> a left) (< a right) (> a right))  )  (defun make\_select\_between (lst left right)  (reduce  #'(lambda (res el)  (cond  ((and (numberp el) (is\_between left right el)) (append res (cons el Nil)))  ((listp el) (append res (make\_select\_between el left right)))  (t res)  )  )lst :initial-value nil  )  ) |

2. Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов. (Напомним, что А х В это множество всевозможных пар (a b), где а принадлежит А, принадлежит В.)

**А) Реализация с использованием функционалов**

|  |
| --- |
| (defun decart (lstx lsty)  (mapcar #'(lambda (x)  (mapcar #'(lambda (y) (list x y)) lsty)) lstx)  ) |

**Б)** **Рекурсивная реализация**

|  |
| --- |
| (defun dec\_rec (el lst)  ( cond  ((null lst) Nil)  (t (cons (cons el (cons (car lst) nil)) (dec\_rec el (cdr lst))) )  )  )  (defun decart\_rec (a b)  ( cond  ((null a) nil)  (t (nconc (dec\_rec (car a) b) (decart\_rec (cdr a) b)))  )  ) |

**3. Почему так реализовано reduce, в чем причина?**

(reduce #'+ ()) -> 0

(reduce #'\* ())-> 1

Сначала функция проверяет список-аргумент. Если он пуст, возвращается значение функции при отсутствии аргументов.

Также reduce использует аргумент :initial-value. Этот аргумент определяет значение, к которому будет применена функция при обработке первого элемента списка-аргумента. Если список-аргумент пуст, то будет возвращено значение initial-value.

Результатом вычисления функции + без аргументов будет 0, а результатом вычисления функции \* без аргументов будет 1, т.к. это нейтральные элементы для данных операций.

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.

|  |
| --- |
| (defun sum\_length (lst)  (reduce #'+ (mapcar #'(lambda (x) (length x)) lst))  ) |

Рекурсия:

|  |
| --- |
| (defun sum\_of\_len\_r(lst res)  ( cond  ((null lst) res)  ((listp (car lst)) (sum\_of\_len\_r (cdr lst) (+ res (length (car lst)))))  (t (sum\_of\_len\_r (cdr lst) (+ res 1)))  )  )  (defun sum\_of\_len(lst)  (sum\_of\_len\_r lst 0)  ) |

**Ответы на теоретические вопросы**

**Классификация рекурсивных функций**

**Рекурсия** — это ссылка на определяемый объект во время его определения.

В LISP существует классификация рекурсивных функций:

● простая рекурсия - один рекурсивный вызов в теле

● рекурсия первого порядка - рекурсивный вызов встречается несколько раз

● взаимная рекурсия - используется несколько функций, рекурсивно вызывающих друг друга.

**Виды рекурсии:**

● Хвостовая. Результат формируется не на выходе из рекурсии, а на входе в рекурсию, все действия выполняются до ухода на следующий шаг рекурсии.

● Дополняемая. При обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова , а вне его.

● Множественная. На одной ветке происходит сразу несколько рекурсивных вызовов. Количество условий выхода также может зависеть от задачи.

● Взаимная.

● Рекурсии высокого порядка. При организации рекурсии можно использовать как функции с именем, так и локально определенные с помощью лямбда выражений функции. Кроме этого, при организации рекурсии можно использовать функционалы или использовать рекурсивную функцию внутри функционала.