|  |  |
| --- | --- |
| **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 № 10**

|  |  |
| --- | --- |
| **Тема** Рекурсивные функции  **Дисциплина** Функциональное и логическое программирование  **Студент** Потапчук А.А.  **Группа** ИУ7-63Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Строганов Ю.В. |  |

Москва.

2020 г.

**Цель работы:** приобрести навыки организации рекурсии в Lisp

**Задачи работы:** изучить способы организации хвостовой, дополняемой, множественной, взаимной рекурсии и рекурсии более высокого порядка в Lisp.

**Теоретическая часть:**

1. Способы организации повторных вычислений в Lisp,
2. Что такое рекурсия? Классификация рекурсивных функций в Lisp,
3. Различные способы организации рекурсивных функций и порядок их реализации,
4. Способы повышения эффективности реализации рекурсии.

*Ответы:*

1. *Функционалы и рекурсия.*
2. *Рекурсия – ссылка на описываемый объект во время его описания.*
3. *Виды рекурсии (по прочинанной мной лекции):* 
   1. *Хвостовая рекурсия – результат формируется на входе в функцию.*
   2. *Рекурсия по нескольким параетрам – результат формируется на основе нескольких параметров.*
   3. *Дополняемая рекурсия – при обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова, а вне его.* ***(Вопрос: правда что дополняемая рекурсия не является хвостовой? Я делаю такой вывод потому, что в этом виде рекурсии нельзя освободить фрейм вызова со стека, так как ещё не все вычисления выполнены. Но думаю, что это можно как-то оптимизировать, чтобы она преобразовывалась в хвостовую, поэтому и обращась к вам.)***
   4. *Множественная рекурсия – на одной ветке происходит сразу несколько рекурсивных вызовов.*
4. *Самый эффективный вид рекурсии – хвостовая. К ней можно привести дополяемую рекурсию посредством добавления к функции дополнительных параметров. При этом нужно создать функцию-оболочку для вызова функции БЕЗ дополнительных параметров.*

**Практическая часть:**

7. 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 (lol – list-of-lists):*

*; добавление числа add к сумме длин элементов списка lol*

*(defun add-lol-length(add lol)*

*(cond*

*((null lol) add)*

*(T (add-lol-length*

*(+ add (length (car lol)))*

*(cdr lol)*

*)*

*)*

*)*

*)*

*; lol - list-of-lists*

*(defun lol-length (lol)*

*(cond*

*((null lol) 0)*

*(T (add-lol-length 0 lol))*

*)*

*)*

8. Написать рекурсивную версию (с именем reg-add) вычисления суммы чисел

заданного списка.

Например: (reg-add (2 4 6)) -> 12

*Решение:*

*; добавлние числа add к сумме чисел списка num-lst*

*(defun num-add(add num-lst)*

*(cond*

*((null num-lst) add)*

*(T (num-add*

*(+ add (car num-lst))*

*(cdr num-lst)*

*)*

*)*

*)*

*)*

*(defun reg-add(num-lst)*

*(cond*

*((null num-lst) 0)*

*(T (num-add 0 num-lst))*

*)*

*)*

9. Написать рекурсивную версию с именем recnth функции nth.

*Решение:*

*(defun recnth(lst n)*

*(cond*

*((null lst) nil)*

*((= n 0) (car lst))*

*(t (recnth (cdr lst) (- n 1)))*

*)*

*)*

10. Написать рекурсивную функцию alloddr, которая возвращает t когда все

элементы списка нечетные.

*Решение:*

*(defun alloddr(lst)*

*(cond*

*((null lst) T)*

*((oddp (car lst)) (alloddr (cdr lst)))*

*(T nil)*

*)*

*)*

11. Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка.

*Решение:*

*(defun reclast(lst)*

*(cond*

*((null (cdr lst)) lst)*

*(T (reclast (cdr lst)))*

*)*

*)*

12. 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 п-аргумента функции до последнего >= 0,

2) от start-аргумента функции до end-аргумента с шагом step.

*Решение (реализован второй вариант, так как я имею 20-ый номер в журнале):*

*(defun step-sum(start end step)*

*(cond*

*((> start end) 0)*

*(T (+ start (step-sum*

*(+ start step)*

*end*

*step*

*)*

*)*

*)*

*)*

*)*

13. Написать рекурсивную функцию, которая возвращает последнее нечетное

число из числового списка, возможно создавая некоторые вспомогательные функции.

*Решение:*

*; Ф-я, которая тянет за собой последний нечетный элемент*

*(defun last-odd-saved-last(lst save-odd)*

*(cond*

*((null lst) save-odd)*

*((oddp (car lst)) (last-odd-saved-last*

*(cdr lst)*

*(car lst)*

*)*

*)*

*(T (last-odd-saved-last*

*(cdr lst)*

*save-odd*

*)*

*)*

*)*

*)*

*; Итоговая ф-я*

*(defun last-odd(lst) (last-odd-saved-last lst nil))*

14. 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 sqr(x) (\* x x))*

*(defun squares(num-lst)*

*(cond*

*((null num-lst) nil)*

*(T (cons*

*(sqr (car num-lst))*

*(squares (cdr num-lst))*

*)*

*)*

*)*

*)*

15. 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. )**

*Решение (вариант 2):*

*; ф-я, запомянающая текущую сумму нечётных чисел*

*(defun odd-add(add num-lst)*

*(cond*

*((null num-lst) add)*

*((odd (car num-lst)) (odd-add*

*(+ add (car num-lst))*

*(cdr num-lst)*

*)*

*)*

*(T (odd-add*

*add*

*(cdr num-lst)*

*)*

*)*

*)*

*)*

*(defun odd-sum(num-lst)*

*(cond*

*((null num-lst) 0)*

*(T (odd-add 0 num-lst))*

*)*

*)*