|  |  |
| --- | --- |
| 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**

**по курсу “Функциональное и логическое программирование”**

**по теме “Рекурсии”**

|  |  |
| --- | --- |
| Студент: | Уласик Е.А. |
| Группа: | ИУ7-61 |
| Преподаватель: | Толпинская Н.Б. |

*2020 г.*Практическая часть

Задание 1. Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list.

(defun sum\_lengths (list-of-lists)

(reduce #'+

(mapcar (lambda (x)

(cond ((listp x) (sum\_lengths x))

(T 1))) list-of-lists

)

)

)

Описание параметров:

list-of-lists – список элементов;

Формирование окончательного результата:

Производится перебор элементов списка list-of-lists. Если встречается элемент-список, то для него вызывается рекурсивная функция, иначе прибавляем к сумме 1.

Тесты:

1. (sum\_lengths '((1 2) () (3 4 5) () (1)))

6

1. (sum\_lengths '((1 2) () (3 (4 3 5)) () (1)))

7

1. (sum\_lengths '((1 2) () 3 4 (3 (4 3 5)) () (1)))

9

1. (sum\_lengths '())

0

Задание 2. Написать рекурсивную версию (с именем reg-add) вычисления суммы чисел заданного списка.

(defun add\_inner (lst sum)

(cond

((null lst) sum)

((listp (car lst)) (add\_inner (cdr lst) (add\_inner (car lst) sum)) )

((numberp (car lst)) (add\_inner (cdr lst) (+ sum (car lst))) )

(T (add\_inner (cdr lst) sum))

)

)

(defun reg-add (lst)

(cond ((eq lst nil) nil)

(T (add\_inner lst 0))

)

)

Описание переменных:

lst – список элементов;

sum – сумма длин;

Формирование окончательного результата:

Функция reg-add проверяет не пуст ли список элементов. Если пуст, то возвращает nil, иначе вызывает функцию add\_inner.

Если функция add\_inner получила пустой список, то возвращает текущую вычисленную сумму; если голова текущего списка – список, то производится рекурсивный вызов add\_inner с параметрами в виде хвоста текущего списка и суммы элементов головы текущего списка, которая также вычисляется рекурсивно; если голова текущего списка – число, то производится рекурсивный вызов add\_inner с параметрами в виде хвоста текущего списка и суммы сложенной со значением головы текущего списка.

Тесты:

1. (rec\_add '(1 2 3 4 5 ( 1 2 3)))

21

1. (add '(1 2 (3 (4)) 5))

15

1. (add '((1 2) (3 4)))

10

1. (add '())

NIL

1. (add '(()))

0

Задание 3. Написать рекурсивную версию с именем recnth функции nth.

(defun rec\_nth\_inner (lst curr target)

(cond ((null lst) nil)

((eq curr target) (car lst))

(t (rec\_nth\_inner (cdr lst) (+ curr 1) target))

)

)

(defun rec\_nth (num lst)

(rec\_nth\_inner lst 0 num)

)

Описание параметров:

num – индекс элемента;

lst – список элементов;

Формирование окончательного результата:

Функция rec\_nth производит вызов функции rec\_nth\_inner с индексом 0.

Если в функцию rec\_nth\_inner передан пустой список, то возвращается nil; если текущий индекс равен индексу искомого элемента, то возвращается голова текущего списка, иначе производится рекурсивный вызов с параметрами в виде хвоста списка, текущего индекса, увеличенного на единицу, и индекса искомого элемента. Если индекс искомого элемента выходит за границу списка, то рекурсия войдёт в rec\_nth\_inner в виде пустого списка и как результат вернётся nil.

Тесты:

1. (rec\_nth 3 '(a f b e q g))

E

1. (rec\_nth 0 '(a b c))

A

1. (rec\_nth 10 '(a b c))

NIL

1. (rec\_nth 1 '(a b))

B

1. (rec\_nth 2 '((a b) c (e t) f))

(E T)

Задание 4. Написать рекурсивную функцию alloddr, которая возвращает t когда все элементы списка нечетные.

(defun alloddr (lst)

(cond

((null lst) t)

((listp (car lst))

(and (alloddr (car lst)) (alloddr (cdr lst))))

((not (numberp (car lst))) nil)

((evenp (car lst)) nil)

(t (alloddr (cdr lst)))

)

)

Описание параметров:

lst – список элементов;

Формирование окончательного результата:

Если функция alloddr принимается в качестве параметра пустой список, то возвращается T; если голова списка – это список, то производится рекурсивный вызов функции alloddr для головы списка и для хвоста; если голова списка – не число и не точечная пара, то возвращается Nil; если голова текущего списка – это чётное число, то возвращается Nil; иначе вызывается рекурсивная функция alloddr для хвоста текущего списка.

Тесты:

1. (alloddr '(1 3 5 7))

T

1. (alloddr '(1 3 5 6))

NIL

1. (alloddr '(1 3 (6) 3))

NIL

1. (alloddr '(1 0 3))
2. Nil

(alloddr '(1 0 a))

NIL

1. (alloddr '((1 3 5 3) 7 4))

NIL

1. (alloddr '((1 3 5 3) 7 5))

T

Задание 5. Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка - аргументы.

(defun my\_last (lst)

(if (eq (cdr lst) nil)

(car lst)

(my\_last (cdr lst))

)

)

Описание параметров:

lst – список элементов;

Формирование окончательного результата:

Если при вызове функции my\_last хвост переданного списка – это Nil, то возвращается голова переданного списка, иначе производится рекурсивный вызов функции my\_last, в который передаётся хвост текущего списка.

Тесты:

1. (my\_last '(1 2 3))

3

1. (my\_last '(a b (c d)))

(C D)

1. (my\_last '(a))

A

1. (my\_last '(a b (c d) e))

E

1. (my\_last '())

NIL

Задание 6. Написать рекурсивную функцию, относящуюся к дополняемой рекурсии с одним тестом завершения, которая вычисляет сумму всех чисел от 0 до n-ого аргумента функции.

(defun sum\_n (lst n)

(if (or (eq lst nil) (eq n 0))

0

(+ (car lst) (sum\_n (cdr lst) (- n 1)))

)

)

Описание параметры:

lst – список элементов;

n – количество слагаемых слева;

Формирование окончательного ответа:

Если переданный список пуст или количество слагаемых равно нулю, то возвращается ноль, иначе суммирует голову списка и результат рекурсивного вызова функции sum\_n с параметрами в виде хвоста списка и количества слагаемых, из которого вычтена 1.

Тесты:

1. (get\_n\_sum '(1 2 3 4 5) 3)

6

1. (get\_n\_sum '(1 2 3) 5)

6

1. (get\_n\_sum '() 0)

0

1. (get\_n\_sum '(1 2 3 4 5) 5)

15

1. (get\_n\_sum '(1 2 3 4 5) 0)

0

Задание 7. Написать рекурсивную функцию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции.

(defun last\_odd\_inner (lst value)

(cond ((eq lst nil) value)

((oddp (car lst)) (last\_odd\_inner (cdr lst) (car lst)))

(t (last\_odd\_inner (cdr lst) value))

)

)

(defun last\_odd (lst)

(last\_odd\_inner lst nil)

)

Описание параметры:

lst – список элементов;

Формирование окончательного ответа:

Функция last\_odd производит вызов функции last\_odd\_inner с параметрами в виде списка lst и nil – последнего нечётного элемента.

Если функция last\_odd\_inner принимает в качестве параметра пустой список, то возвращается последнее найденное нечётное число; если голова текущего списка – это нечётное число, то производится рекурсивный вызов функции last\_odd\_inner с параметрами в виде lst и головы текущего списка; иначе производится рекурсивный вызов с параметрами в виде хвоста текущего списка и последнего найденного нечётного числа.

Тесты:

1. (last\_odd '(1 2 3 4))

3

1. (last\_odd '(3))

3

1. (last\_odd '())

NIL

1. (last\_odd '(2 4 6 8))

NIL

1. (last\_odd '(1 4 6 8))

1

1. (last\_odd '(1 4 6 8 15))

15

Задание 8. 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\_all (lst)

(cond

((null lst) nil)

(t (cons (\* (car lst) (car lst)) (square\_all (cdr lst))))

)

)

Принимаемые параметры:

lst – список элементов;

Формирование окончательного результата:

Если функция принимает пустой список, то возвращается Nil, иначе с помощью функции cons создаётся списочная ячейка, car указатель которой указывает на квадрат значения переданного списка, а cdt указывает на результат вызова рекурсивной функции square\_all с параметрами в виде хвоста переданного списка.

Тесты:

1. (square\_all '(1 2 3))

(1 4 9)

1. (square\_all '(0))

(0)

1. (square\_all '())

NIL

1. (square\_all '(1 2 3 4 -3 16))

(1 4 9 16 9 256)

Задание 9. Написать функцию с именем select-odd, которая из заданного

списка выбирает все нечетные числа.

(defun select-odd-rec (lst)

(cond

((null lst) nil)

(T (cons (cond ((oddp (car lst)) (car lst)) (T Nil)) (select-odd-rec (cdr lst))))

)

)

(defun select-odd (lst)

(remove nil (select-odd-recur lst))

)

Принимаемые параметры:

lst – список элементов;

Формирование окончание результат:

Функция select-odd вызывает функцию select-odd-rec с параметром lst.

Если в select-odd-rec передан пустой список, то вернётся nil, иначе создаётся списочная ячейка, car указатель которой указывает на голову текущего списка, если эта голова – чётное число, иначе на nil, а cdr-указатель – на хвост результат вызова рекурсивной функции select-odd-rec с параметром в виде хвоста текущего списка.

Тесты:

1. (select-odd '(1 4 1 3 6))

(1 1 3)

1. (select-odd '(1 1 3))

(1 1 3)

1. (select-odd '(2 4 6))

NIL

1. (select-odd '(1))

(1)

1. (select-odd '(-1 -3))

(-1 -3)

1. (select-odd '(0))

NIL

1. (select-odd '())

NIL

**Теоретические вопросы:**

* Способы организации повторных вычислений в Lisp.

Для организации повторных вычислений в Lisp могут быть использованы функционалы – функции высокого порядка, которые получают в качестве аргумента функциональный объект. Также для организации повторных вычислений может быть использована рекурсия. Рекурсия – это ссылка на определяемый объект во время его определения.

* Различные способы использования функционалов.

Функционалы используются в Lisp для организации повторных вычислений. В Lisp используются применяющие и отображающие функционалы, функционалы, являющиеся предикатами, функционалы, использующие предикаты в качестве функционального объекта.

* Что такое рекурсия? Способы организации рекурсивных функций.

Рекурсия – это ссылка на определяемый объект во время его определения. Виды рекурсии: хвостовая, дополняемая, множественная, взаимная и рекурсии высокого порядка.

* Способы повышения эффективности реализации рекурсии.

Один из методов повышения эффективности рекурсии является организация хвостовой рекурсии. Для этого может потребоваться использовать дополнительные параметры. Такая рекурсия может быть путём формальной и гарантированно корректной перестройки кода заменена на итерацию. Такая оптимизация реализована во многих оптимизирующих компиляторах, а в трансляторах Scheme, одного из диалектов Lisp, такая оптимизация является обязательной.