**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

**«**Числовые функции языка Лисп**»**

Выполнил: студент гр. ИТП-31

Коркуц С. И.

Принял: профессор

Мурашко И. А.

Гомель 2020

**Цель работы:** закрепление навыков работы с функциями Лиспа.

**Ход выполнения работы**

1. Вычислить среднее арифметическое отрицательных чисел произвольного списка.

**Алгоритм решения:**

1. Проверка на пустоту
2. Проверка аргумента на лист
3. Проверка на отриц. число
4. Сумма отриц. значений в списке и сумма их кол-ва
5. Рекурсия с оставшимся списком
6. Соединение в один лист (сумма первого элемента списка с первым элементом второго списка)
7. Проверка кол-во отриц.значений, если да, то возврат деления суммы отрицательных значений на их кол-во
8. Если нет, возврат 0

**Листинг:**

(defun negativeMean (lst)

;(sumNegative lst)

(meanList (sumNegative lst))

)

(defun sumNegative (lst)

(cond

((Null lst),

(cons 0 0)

)

((listp (car lst))

(calcLists (sumNegative (car lst)) (sumNegative (cdr lst)))

)

((< (car lst) 0)

(calcLists (cons (car lst) 1) (sumNegative (cdr lst)))

)

(

(calcLists (cons 0 0) (sumNegative (cdr lst)))

)

)

)

(defun calcLists (lst1 lst2)

(cons (+ (car lst1) (car lst2)) (+ (cdr lst1) (cdr lst2)))

(defun meanList (lst)

(cond

((> (cdr lst) 0)

(/ (car lst) (cdr lst))

)

(

0

)

)

)

**Результат работы:**

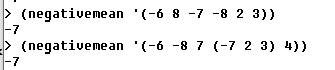


Рисунок 1 – Результат работы программы

**2,3,4.** Представить целое N в виде суммы квадратов двух, трёх, четырёх целых чисел.

**Алгоритм решения:**

1. Проверка входного числа, больше ли оно, чем квадрат по циклу(i++)
2. Проверка входного числа, равно ли оно квадрату полученного I, если да то возвращаем i
3. Если нет, возврат -1
4. Проверка n на 1, если да, вызов функции sqrt
5. Проверка, если i^2 <= x/n, то вызов функции search, находя значения суммы квадратов разных чисел, выполняющих условие, сумма которых будет ли равна x
6. Если значение нашей функции f2-1, то прибавляем к i++, иначе вывод полученной записи

**Листинг:**

(defun F (x n)

(F2 x n 1)

)

(defun F2 (x n i)

(cond

( (eq n 1) (list (sqrt x 1)) )

((<= (\* i i) (/ x n))

(search x (F2 (- x (\* i i)) (- n 1) i) i n))

(t (list -1))

)

)

(defun search (x res i n)

(cond

((eq (car res) -1) (F2 x n (+ i 1)))

(t (cons i res))

)

)

(defun sqrt (x i)

(cond

((< (\* i i) x) (sqrt x (+ i 1)))

((= (\* i i) x) i)

(t -1)

)

)

**Результат работы:**

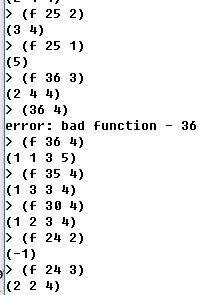


Рисунок 2 – Результат работы программы

**5.** Определить предикат, который проверяет, является ли произвольный список монотонной последовательностью чисел, или нет (без учета скобок).

**Листинг:**

(defun is-mon (lst)

(let ((a1 (apply '<= lst))

(a2 (apply '>= lst)))

(or a1 a2)))

(defun checkMonotonousRec (lst)

(is-mon (openList lst))

)

(defun openList (lst)

(cond

((Null lst)

Nil

)

((listp (car lst))

(append (openList (car lst)) (openList (cdr lst)))

)

(

(cons (car lst) (openList (cdr lst)))

)

)

)

**Результат работы:**

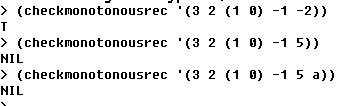


Рисунок 3 – Результат работы программы

**6.** Найти максимальную глубину вложенности произвольного списка.

**Листинг:**

(defun depth (lst)

(cond

((atom lst) 0)

((null lst) 0)

(T (max

(+ 1 (depth (car lst)))

(depth (cdr lst))

))))

**Результат работы:**

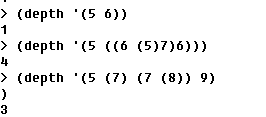


Рисунок 4 – Результат работы программы

**7.** Используя управляющую конструкцию **DO**, вычислить среднее арифметическое чисел 0.1+0.25+…+15.1.

**Алгоритм решения:**

1. Находим i, т.е. сколько всего чисел получилось
2. Идем по циклу, прибавляя 0,15, и суммируем
3. Сумму делим на i

**Листинг:**

(defun q ()

(setq sum 0)

(setq i (/ (- 15.1 0.1) 0.15))

(do

( (cur 0.1 (+ cur 0.15) ) )

( (> cur 15.25) (/ sum i ) )

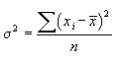
(setq sum (+ sum cur ))))

**Результат работы:**



Рисунок 5 – Результат работы программы

**8.** Используя управляющую конструкцию **DOlist**, вычислить среднее арифметическое и дисперсию чисел из простого списка.



**Алгоритм решения:**

1. Объявление переменных
2. Foreach, где сложение кол-ва и суммы значений
3. Вычисление среднего значения
4. Идем по списку, где идёт сложение текущего x минус среднее значение
5. Вычисление дисперсии

**Листинг:**

(defun dispAndMean (lst)

(let ((res 0) (mn 0) (i 0))

(dolist (x lst mn)

(setq i (+ i 1))

(setq mn (+ mn x))

)

(setq mn (/ mn i))

(dolist (x lst res)

(setq res (+ res (square (- x mn))))

)

(cons (/ res i) mn)

)

)

(defun square (x)

(\* x x)

)

**Результат работы:**

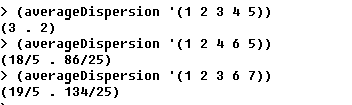


Рисунок 6 – Результат работы программы

**9.** Даны два списка вида (a1,a2,…) и (b1,b2,…). Используя отображающий функционал MAPx, получить список вида (a1,b1,a2,b2…). Если исходные списки разной длины, то остаток более длинного списка дописать в конец без изменения.

**Листинг:**

(defun f9 (\_list1 \_list2)

(mapcon #'f \_list1 \_list2)

)

(defun f (\_list1 \_list2)

(cond

((null (cdr \_list1)) (cons (car \_list1) \_list2))

((null (cdr \_list2)) (cons (car \_list1) (cons (car \_list2) (cdr \_list1))))

(t (list (car \_list1) (car \_list2)))

)

)

**Результат работы:**

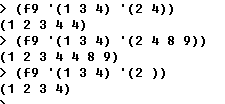


Рисунок 7 – Результат работы программы

**10.** Разработать функцию преобразования и вычисления математических выражений в префиксную форму.

**Листинг:**

(defun solve (\_list)

(f1 (f3 \_list nil nil))

)

(defun f1 (\_list)

(cons \_list (eval \_list))

)

(defun f2 (x)

(cond

((or (eq x '/) (eq x '\*)) 2)

((or (eq x '+) (eq x '-)) 1)

(t nil)

)

)

(defun f3 (\_list vlist olist)

(cond

((null \_list) (f6 vlist olist))

((f2 (car \_list)) (f5 (car \_list) \_list vlist olist))

((listp (car \_list)) (f3 (cdr \_list) (cons (f3 (car \_list) nil nil) vlist) olist))

(t (f3 (cdr \_list) (cons (car \_list) vlist) olist))

)

)

(defun f4 (vlist olist)

(list (car olist) (car (cdr vlist)) (car vlist))

)

(defun f5 (operator \_list vlist olist)

(cond

((null olist) (f3 (cdr \_list) vlist (list operator)))

((<= (f2 operator) (f2 (car olist))) (f3 (cdr \_list) (cons (f4 vlist olist) (cdr (cdr vlist))) (cons operator (cdr olist))))

(t (f3 (cdr \_list) vlist (cons operator olist)))

)

)

(defun f6 (vlist olist)

(cond

((null (cdr olist)) (f4 vlist olist))

(t (f6 (cons (f4 vlist olist) (cdr (cdr vlist))) (cdr olist)))

)

)

**Результат работы:**

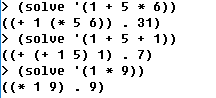


Рисунок 8 – Результат работы программы

**Вывод**:В данной лабораторной работе были закреплены навыки работы с LISP.