Для решения поставленных задач был использован диалект языка Лисп – newLISP.

**Задача №1 (Lab4-1.lsp):**

Факториал числа n – это произведение натуральных чисел от 1 до n. Формула факториала:

Идея в реализованных функциях состоит в том, что перемножаются числа от заданного числа n до 1. Изначально первым множителем становится число n, далее число уменьшается на 1 и передается в эту же функцию, пока n не станет равно 1. Отличие двух функций заключается в использование различных локальных определений LET и LAMBDA, разница которых заключается в том что формальные и фактические параметры LET-вызова помещены совместно в начале формы.

Тестовые наборы данных и результаты (так как результаты выполнения двух функций одинаковы, то поэтому они приведены в одном столбце *Результат*) приведены в Таблице 1:

Таблица 1. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые данные | Результат |
| 1 | 1 |
| 3 | 6 |
| 5 | 120 |

**Задача №2 (Lab4-2.lsp):**

Символьным дифференцирование в математике называется операция преобразования одного арифметического выражение, которое называется производной. Пусть U и V обозначает арифметическое выражение, которое может содержать переменную x, c – константа.

Правила преобразования приведены в Таблице 2:

Таблица 2. Тестовые набора данных

|  |  |
| --- | --- |
| Исходное выражение | Результат |
| с | 1 |
| x | 1 |
| U + V | (+ U’ V’) |
| U - V | (- U’ V’) |
| сU | (\* c U’) |
| U \* V | (+ (\* U V’) (\* V U’)) |
| U / V | (/ (- (\* U’ V) (V’ U)) (^ V 2)) |
|  | (\* c (^ U (- c 1))) |
| lnU | (\* (^ U -1) U’) |

Тестовые наборы данных и результаты приведены в Таблице 3:

Таблица 3. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые данные | Результат |
| '(3) | 0 |
| '(x) | 1 |
| '(^ x 4) | (\* 4 (^ x 3)) |
| '(+ (x) (^ x 3)) | (+ 1 (\* 3 (^ x 2))) |
| '(- (\* 3 x) (^ x 3)) | (- 3 (\* 3 (^ x 2))) |
| '(\* 3 x) | 3 |
| '(\* 3 (^ x 2)) | (\* 3 (\* 2 (^ x 1))) |
| '(\* (x) (^ x 2)) | (+ (\* 1 (^ x 2)) (\* (\* 2 (^ x 1)) (x))) |
| '(/ (\* 3 x) (^ x 3)) | (/ (- (\* 3 (^ x 3)) (\* (\* 3 (^ x 2)) (\* 3 x))) (^ (^ x 3) 2)) |
| '(ln (x)) | (\* (^ (x) -1) 1) |
| '(ln (^ x 5)) | (\* (^ (^ x 5) -1) (\* 5 (^ x 4))) |

**Задача №3 (Lab4-3-1.lsp):**

Идея решения данного задания не отличается от решения в Лабораторной работе №2, однако теперь с помощью локальных определений мы храним голову списка в формальном параметре.

Условие (null? lst) выполняется в нескольких случаях: когда изначально на вход подается пустой список или в ходе работы рекурсии он приходит пустым. Выполнение этого условия означает, что в исходном списке нет атомарного элемента под необходимом номером.

Условие ((and (= n 1) (atom? x)) выполняется если голова текущего списка (формальный параметр x) является атомарным элементом и n = 1 (значение n в ходе рекурсии уменьшается поэтому если n = 1, то это требуемый для решения атомарный элемент). Выполнение этого условия означает, что необходимый по заданию атомарный элемент был найден.

Условие (atom? x) выполняется если голова текущего списка (формальный параметр x) является атомарным элементом, при этом функция вызывает сама себя уменьшив значение n на 1 и вместо целого списка передает его хвост.

Если же не выполняются 3 предыдущих условия, то функция вызывает сама себя не изменяя значение n и вместо целого списка передает его хвост.

Тестовые наборы данных и результаты (так как результаты выполнения двух функций одинаковы, то поэтому они приведены в одном столбце *Результат*) приведены в Таблице 4:

Таблица 4. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые наборы данных | Результат |
| ‘ ((2) (3) 4 5 a (e r) g) 3 | A |
| ‘() 2 | nil |
| ‘ ((2) (3) (e r)) 3 | nil |
| ‘(b) 1 | b |
| ‘ ((2) (3) 4 (e r) g) 1 | 4 |

**Задача №4–5 (Lab4-4(5).lsp):**

В реализованном простейшим интерпретаторе были реализованы такие базовые функции Лиспа как FIRST (функция my-first), REST (функция my-rest), CONS (функция my-cons) STRING (функция my-string), а также такие арифметические операции как + (функция addition), - (функция subtraction), \* (функция multiplication) и / (функция division).

При вызове функции lisp-interpreter (выполняет роль простейшего интерпретатора) вызывается функция function-definition (определение функции), если данная функция была реализована, то выполняется соответствующая ей функция, иначе мы выводим весь список.

Тестовые наборы данных и результаты приведены в Таблице 5:

Таблица 5. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые наборы данных | Результат |
| '(+ (+ 1 2)(+ 12 8))) | 23 |
| '(- (+ 18 2)(- 12 8)) | 16 |
| '(\* (\* 2 2)(\* 10 2)) | 80 |
| '(/ (/ 30 5)(/ 6 2)) | 2 |
| '(- (+ (/ 25 5)(\* 6 2))(\* 3 2)) | 11 |
| '(first (15 2 3 4)) | 15 |
| '(rest (15 2 3 4)) | (2 3 4) |
| '(cons a 15 1) | (a 15 1) |
| '(first (rest (13 4 2 6 3))) | 4 |
| '(string "Hello, " "world!") | "Hello, world!" |

**Задача №6 (Lab4-3-2.lsp):**

Для начала происходит проверка на то, что количество элементов в списке меньше ли чем n, если же это условие верно, то возвращается весь список (в условие задания не указано что требуется делать в этом случае).

Далее рекурсивно составляется список (функцией CONS): берется голова исходного списка (с помощью локальных определений LET и LAMBDA), после чего вызывается исходная функция при этом в качестве списка передается хвост исходного списка и n уменьшается на 1.

Были реализованы две функции с использованием локальных определений. Одна с использованием LET, а другая – LAMBDA.

Тестовые наборы данных и результаты приведены в Таблице 6:

Таблица 6. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые наборы данных | Результат |
| '() 3 | () |
| '(2 4) 2 | (2 4) |
| '(2 4) 0 | () |
| '(4 3 3 2) 3 | (4 3 3) |
| '(4 3 9 3 8 3 2) 4 | (4 3 9 3) |