Міністерство освіти та науки України

Національний технічний університет України “КПІ ім. Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

**ЗВІТ**

про виконання

Лабораторної роботи №3

З дисципліни:

«**Мультипарадигменне програмування**»

**Тема: «Локальные определения и функционалы»**

**Варіант 1**

**Виконав:** студент групи ІП-51

Булатов Дмитро Эгорович

**Перевірив**: викл.

Баклан Ігор Всеволодович

Київ 2017

**Мета роботи:**

Целью работы является практическое изучение различных видов локальных определений и особенностей их использования в рекурсивных программах.

**Основні задачі:**

* Изучить применение техники нисходящей и восходящей рекурсии при написании рекурсивных функций с использованием локальных определений;
* Сравнить возможности локальных определений LET и LAMBDA по организации вычислений в рекурсивных программах;

**Завдання 1**

Описать функцию вычисления факториала. Рассмотреть варианты решения задачи с применением локальных определений LAMBDA и LET.

**Код програми:**

(defun fact1(A)

(cond

((= A 0) 1)

(T ((lambda (A B) (\* A B)) A (fact1 (- A 1))))

)

)

(print

(fact1 6)

)

(defun fact2(A)

(cond

((= A 0) 1)

(T (let

(

(C A)

(D (fact2 (- A 1)))

)

(\* C D)

))

)

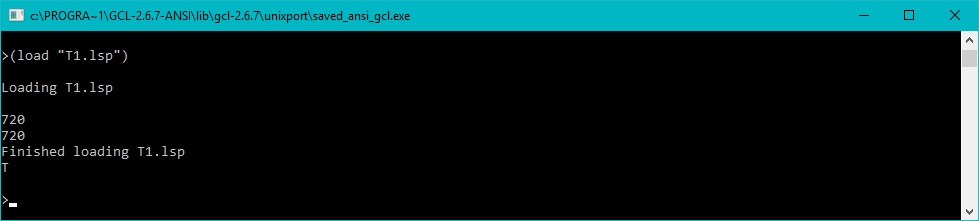
)

(print

(fact2 6)

)

**Скріншот програми:**



В цій програмі я створив 2 функції підрахування факторіалу. Перша викорустовує lambda, а друга let.

**Завдання 2**

Разработать программу символьного дифференцирования в соответствии с правилами, изложенными в [3], стр. 194-196. Рассмотреть варианты решения задачи с применением локальных определений LAMBDA и LET.

**Код програми:**

(defun der (e)

(cond

((null e) 0)

((equal e 'x) 1)

((atom e) 0)

((null (cdr e)) (der (car e)))

((null (cddr e))

(cond

((equal (car e) '+ ) (der (cadr e)))

((equal (car e) '- ) (list '- (der (cadr e))))

(t (derfun (car e) (cadr e))) )

)

(t (derexpr (car e) (cadr e) (caddr e)))

)

)

(defun derexpr (arg1 op arg2 )

(cond

((equal op '+ ) (deradd arg1 arg2 ))

((equal op '- ) (dersub arg1 arg2 ))

((equal op '\* ) (dermult arg1 arg2))

((equal op '/ ) (derdiv arg1 arg2))

((equal op '^ ) (derpower arg1 arg2))

(t (print 'err))

)

)

(defun derfun (fun arg)

(cond

((equal 'SIN fun) (list (list 'COS arg) '\* (der arg) ))

((equal 'COS fun) (list (list '- (list 'SIN arg)) '\*

(der arg) ))

((equal 'EXP fun) (list (list 'EXP (list arg)) '\*

(der arg) ))

((equal 'LOG fun) (list (der arg) '/ arg ))

(t (print 'illegal\_function))

)

)

(defun deradd (arg1 arg2)

(list (der arg1) '+ (der arg2))

)

(defun dersub (arg1 arg2)

(list (der arg1) '- (der arg2))

)

(defun derdiv (arg1 arg2)

(list

(list (list (der arg1) '\* arg2)

'- (list arg1 '\* (der arg2) ))

'/ (list arg2 '^ '2)

)

)

(defun dermult (arg1 arg2)

(list (list (der arg1) '\* arg2)

'+ (list arg1 '\* (der arg2))

)

)

(defun derpower (arg1 arg2)

(list (list arg1 '^ arg2)

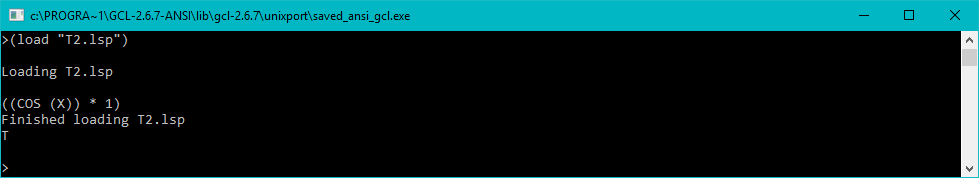
'\* (dermult arg2 (list 'LOG(list arg1)))

)

)

(print(der '(sin (x))))

**Скріншот програми:**

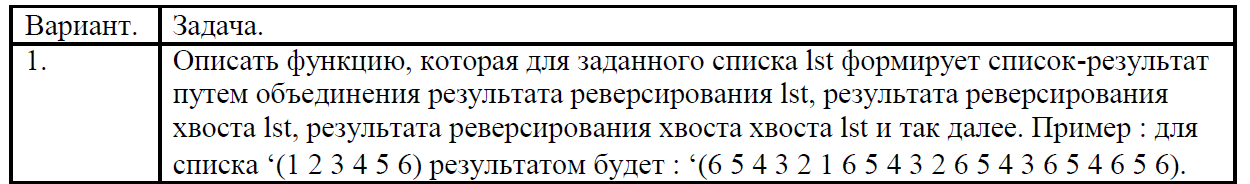


В цьому завданні я розбив программу що виконує диферінцюювання виразу за допомогою певної таблиці правил. Для вирішення задачі я спочатку перевіряю я знаходжу операцію що виконується а потім запускаю відповідну функцію для підрахунку диференціалу.

**Завдання 3**

Решить задачу из лабораторной работы №2 с применением локальных определений

LAMBDA и LET.



**Код програми:**

(defun rev1(A)

(

(lambda (A)

(cond

((atom A) A)

(T (append (rev1 (cdr A)) (list (car A))))

)

) A

)

)

(defun V1(A)

(cond

((atom A) A)

(T (append (rev1 A) (V1 (cdr A))))

)

)

(print

(V1 '(1 2 3 4 5 6))

)

(defun rev2(A)

(let ((B A))

(cond

((atom B) B)

(T (append (rev2 (cdr B)) (list (car B))))

)

)

)

(defun V2(A)

(cond

((atom A) A)

(T (append (rev2 A) (V2 (cdr A))))

)

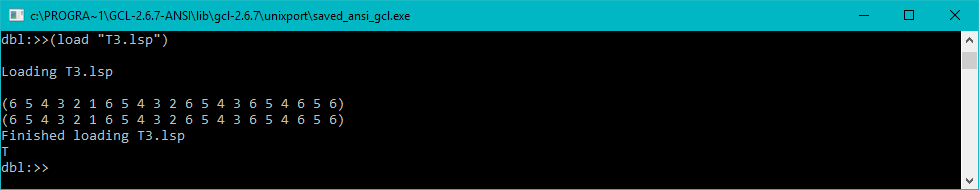
)

(print

(V2 '(1 2 3 4 5 6))

)

**Скріншот програми:**



В цьому завданні я використав той самий алгоритм що у 2й лабораторній але замінив частину коду на виклик локальних визначень через lambda\let

**Завдання 4**

Реализовать программу- простейший интерпретатор лисповских программ . На вход интерпретатора подается текст, который может быть интерпретирован как вызов или суперпозиция функций Лиспа, пример (для GCLisp'а) : '(cons(car(cdr '(e r t w))) (cons (cdr '(g h 6)) nil)). Программа должна обеспечивать выполнение такого рода примеров.

Требования к программе :

− Должна обеспечивать интерпретацию базовых функций Лиспа и арифметических операций +, -, /, \*;

− В программе должны использоваться локальные определения;

− Не допускается использование встроенной функции-интерпретатора EVAL;

**Код програми:**

(defun my\_eval(A)

(cond

((atom A) A)

((equal 'car (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)))))

((equal 'cdr (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)))))

((equal 'atom (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)))))

((equal 'cons (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal 'list (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal 'equal (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '\* (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '/ (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '+ (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '- (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '= (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

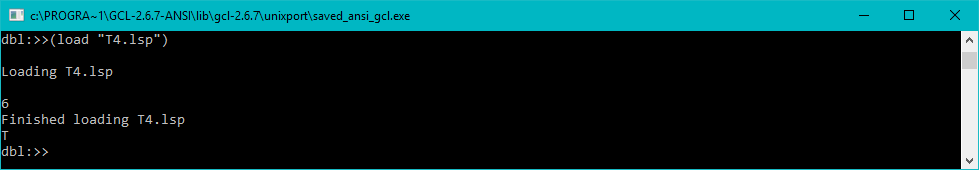
(T A)

)

)

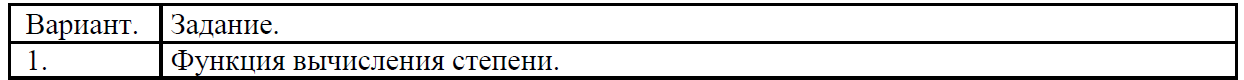
(print (my\_eval '(+ (car (car (cdr (cdr ((1 2) 3 (4 5)))))) (car (cdr (1 2))))))

**Скріншот програми:**



В цьому завданні я написав функцію що перевіряє перший елемент прийнятого списку на те чи він функція, і якщо це так то виконує локально визначену функцію інтерпритації в залежності від кількості аргументів цієї функції, і просто повертає прийняте значення якщо воно не є базовою функцією.

**Завдання 5**

Дополнить интерпретатор из задания 4 в соответствии с вариантом индивидуального задания из Таблицы 1. 

**Код програми:**

(defun pow(A B)

(cond

((= B 0) 1)

((> B 0) (\* A (pow A (- B 1))))

(T (\* (/ 1 A) (pow A (+ B 1))))

)

)

(defun my\_eval(A)

(cond

((atom A) A)

((equal 'car (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)))))

((equal 'cdr (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)))))

((equal 'atom (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)))))

((equal 'cons (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal 'list (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal 'equal (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '\* (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '/ (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '+ (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '- (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '= (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal 'pow (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

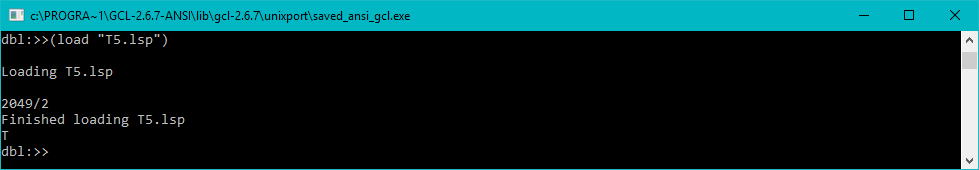
(T A)

)

)

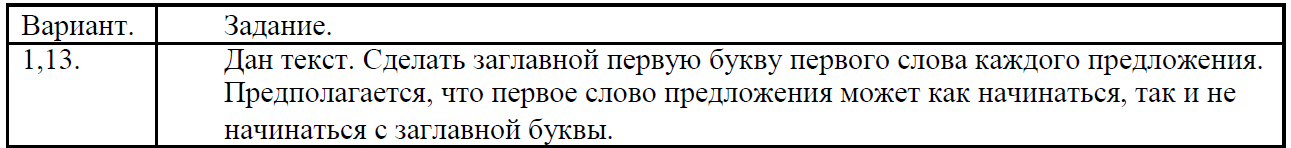
(print (my\_eval '(+ (pow 2 10) (pow 2 -1)))))

**Скріншот програми:**



В цьому завданні я написав функцію підношення числа до данної степені, і включив її до списку функцій що може виконати мій інтерпритатор

**Завдання 6**

Написать программу обработки текста естественного языка с использованием отображающих функционалов в соответствии с заданием из таблицы. Текст рекомендуется представлять списком списков : каждое предложение- список слов, весь текст- список предложений. 

**Код програми:**

(defun pow(A B)

(cond

((= B 0) 1)

((> B 0) (\* A (pow A (- B 1))))

(T (\* (/ 1 A) (pow A (+ B 1))))

)

)

(defun my\_eval(A)

(cond

((atom A) A)

((equal 'car (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)))))

((equal 'cdr (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)))))

((equal 'atom (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)))))

((equal 'cons (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal 'list (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal 'equal (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '\* (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '/ (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '+ (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '- (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal '= (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

((equal 'pow (car A)) (let ((A A)) (funcall (car A) (my\_eval (cadr A)) (my\_eval (caddr A)))))

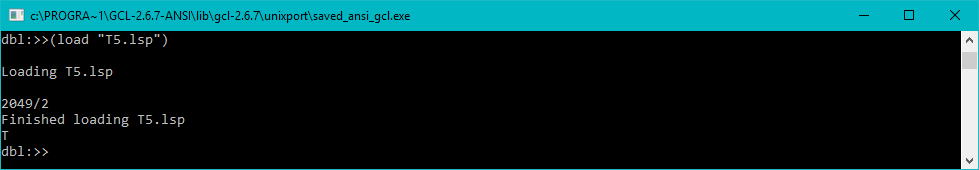
(T A)

)

)

(print (my\_eval '(+ (pow 2 10) (pow 2 -1)))))

**Скріншот програми:**



В цьому завданні я написав функцію підношення числа до данної степені, і включив її до списку функцій що може виконати мій інтерпритатор

**Висновок:**

В цій лабораторній роботі я опанував базовий механізм виклику та опису рекурсивних функцій та рекурсивної обробки списків. Також я навчився писати рекурсивні реалізації алгоритмів сортувань на функціональній мові.