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:

«**Декларативне програмування**»

**Тема: «Локальні визначення та функціонали»**

**Варіант 6**

**Виконав:** студент групи ІП-51

Зарічковий Олександр Анатолійович

**Перевірив**: доц. Баклан І. В.

Київ 2017

**Варіант 6**

**Мета роботи:** практичне вивчення різних видів локальних визначень і особливостей їх використання в рекурсивних програмах.

**Основні завдання:**

* вивчити застосування техніки низхідної і висхідної рекурсії при написанні рекурсивних функцій з використанням локальних визначень.
* порівняти можливості локальних визначень LET і LAMBDA по організації обчислень в рекурсивних програмах.

**Виконання завдань**

***Завдання 1***

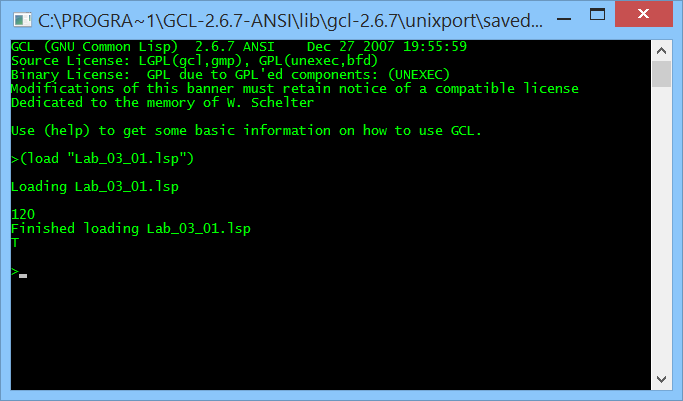
|  |  |
| --- | --- |
| *Варіант* | *Задача* |
| 6 | Описати функцію обчислення факторіала. Розглянути варіанти вирішення завдання з застосуванням локальних визначень LAMBDA і LET. |

*Розв’язання*

Код програми:

|  |
| --- |
| (defun fact (n)  (cond ((equal n 0) 1)  (t ((lambda (x y)(\* x y))  n (fact (- n 1))  )  )  )  )  (print (fact 5)) |

Скріншот виконання програми:



Висновок:

В даній програмі я реалізував функцію факторіалу за допомогою рекурсивної функції fact та локальної анонімної функції перемноження двох чисел, яка обявлена за допомогою ключового слова lambda.

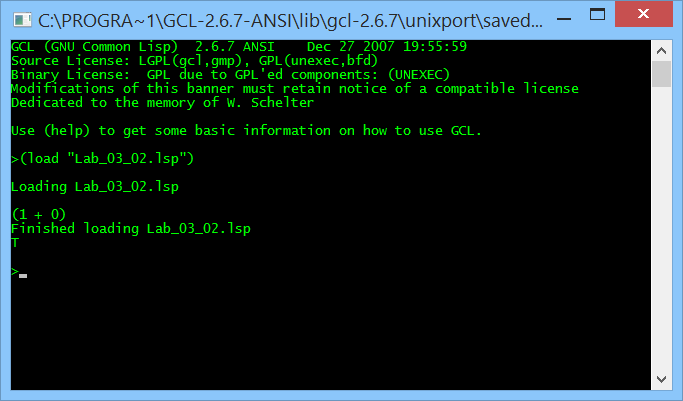
***Завдання 2***

|  |  |
| --- | --- |
| *Варіант* | *Задача* |
| 6 | Розробити програму символьного диференціювання відповідно до правил. Розглянути варіанти вирішення завдання із застосуванням локальних визначень LAMBDA і LET. |

Код програми:

|  |
| --- |
| (defun deriv (e)  (cond ((null e) 0)  ((equal e 'x) 1)  ((atom e) 0)  ((null (cdr e)) (deriv (car e)))  ((null (cddr e)); monadic operator +, -, or function id  (cond ((equal (car e) '+ ) (deriv (cadr e))) ;+  ((equal (car e) '- ) (list '- (deriv (cadr e))));-  (t (derfun (car e) (cadr e))) ) ) ; function  (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 'error))  )  )  (defun derfun (fun arg)  (cond ((equal 'SIN fun) (list (list 'COS arg) '\* (deriv arg) ))  ((equal 'COS fun) (list (list '- (list 'SIN arg)) '\*  (deriv arg) ))  ((equal 'EXP fun) (list (list 'EXP (list arg)) '\*  (deriv arg) ))  ((equal 'LOG fun) (list (deriv arg) '/ arg ))  (t (print 'illegal\_function))  )  )  (defun deradd (arg1 arg2)  (list (deriv arg1) '+ (deriv arg2))  )  (defun dersub (arg1 arg2)  (list (deriv arg1) '- (deriv arg2))  )  (defun derdiv (arg1 arg2)  (list (list (list (deriv arg1) '\* arg2)  '- (list arg1 '\* (deriv arg2) ))  '/ (list arg2 '^ '2)  )  )  (defun dermult (arg1 arg2)  (list (list (deriv arg1) '\* arg2)  '+ (list arg1 '\* (deriv arg2))  )  )  (defun derpower (arg1 arg2)  (list (list arg1 '^ arg2)  '\* (dermult arg2 (list 'LOG(list arg1)))  )  )  (print(deriv '(x + 3))) |

Скріншот виконання програми:



Висновок:

В даній програмі я реалізував функцію символьного дефернціювання описану в завданні лабораторної роботи на мові LISP.

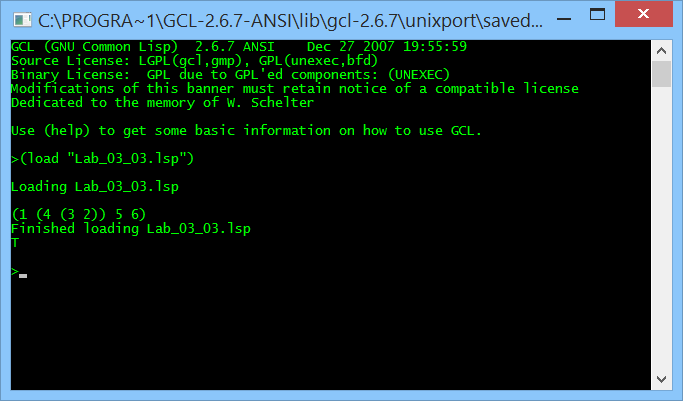
***Завдання 3***

|  |  |
| --- | --- |
| *Варіант* | *Задача* |
| 6 | Вирішити завдання з лабораторної роботи №2 з застосуванням локальних визначень LAMBDA і LET. Реалізація завдання 1. |

Код програми:

|  |
| --- |
| (defun task (lst)  (let ((res nil))  (dolist (i lst (reverse res))  (if (atom i) (push i res) (push (task (reverse i)) res)))  )  )  (print (task '(1 ((2 3) 4) 5 6))) |

Скріншот виконання програми:



Висновок:

В даній програмі я реалізував функцію рекурсивного реверсування усіх вкладених списків на мові LISP.

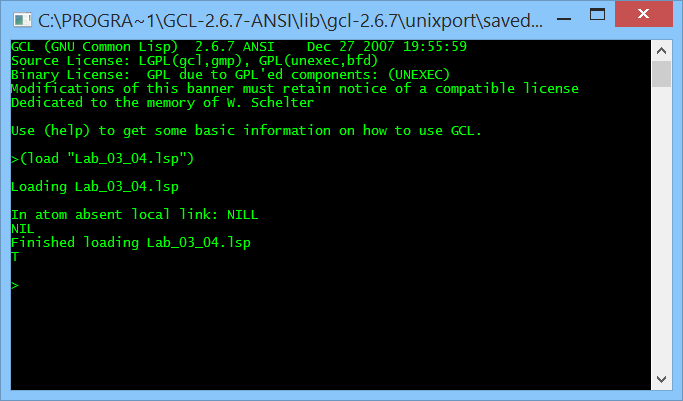
***Завдання 4***

|  |  |
| --- | --- |
| *Варіант* | *Задача* |
| 6 | Реалізувати програму – найпростіший інтерпретатор ліспівських програм. На вхід інтерпретатора подається текст, який може бути інтерпретований як виклик або суперпозиція функцій Ліспу.  *Приклад (для GCLisp'а): '(cons (car (cdr' (e r t w))) (cons (cdr '(g h 6)) nil)).*  Програма повинна забезпечувати виконання такого роду прикладів. |

Код програми:

|  |
| --- |
| (defun my\_eval (loc\_form &optional (loc\_links nil))  (cond  ((atom loc\_form)  (cond  ((eq loc\_form 't1) 't1)  ((eq loc\_form 'nil) 'nil1)  ((numberp loc\_form) loc\_form)  ((car (assoc loc\_form loc\_links)))  (t (format t  "~%In atom absent local link: ~S"  loc\_form))  ((atom (car loc\_form))  (cond  ((eq (car loc\_form) 'quote1)  (cadr loc\_form))  ((eq (car loc\_form) 'cond1)  (eval-cond (cdr loc\_form) loc\_links))  ((get (car loc\_form) 'fn)  (my\_apply (get (car loc\_form) 'fn)  (eval-list (cdr loc\_form)  loc\_links)  loc\_links))  (t (my\_apply (car loc\_form)  (eval-list (cdr loc\_form)  loc\_links)  loc\_links))))  (t (my\_apply (car loc\_form)  (eval-list (cdr loc\_form) loc\_links)  loc\_links)))))  )    (defun eval-cond (branches context)  (cond  ((null branches) 'nil1)  ((not (eq (my\_eval (caar branches) context)  'nil1))  (my\_eval (cadar branches) context))  (t (eval-cond (cdr branches) context)))  )  (defun my\_apply (func arg loc\_links)  (cond ((atom func)  (cond  ((eq func 'car1)  (cond ((eq (car arg) 'nil1)  'nil1)  (t (caar arg))))  ((eq func 'cdr1)  (cond ((eq (car arg) 'nil1)  'nil1)  ((null (cdar arg))  'nil1)  (t (cdar arg))))  ((eq func 'cons1)  (cond ((eq (cadr arg)  'nil1)  (list (car arg)))  (t (cons (car arg)  (cadr arg)))))  ((eq func 'atom1)  (cond ((atom (car arg))  't1)  (t 'nil1)))  ((eq func 'equal1)  (cond ((equal (car arg)  (cadr arg))  't1)  (t 'nil1)))  (t (format t "~%Unkown function:  ~S" func))))  ((eq (car func) 'lambda1)  (my\_eval (caddr func)  (create-links (cadr func)  arg loc\_links)))  (t (format t  "~%It's not lambda call: ~S"  func)))  )  (defun create-links  (loc\_forms params env)  (cond  ((null loc\_forms) env)  (t (acons (car loc\_forms)  (car params)  (create-links (cdr loc\_forms)  (cdr params)  env))))  )  (defun eval-list (params loc\_links)  (cond  ((null params) nill)  (t (cons  (my\_eval (car params) loc\_links)  (eval-list (cdr params)  loc\_links))))  )  (print (my\_eval 'NILL)) |

Скріншот виконання програми:



Висновок:

В даній програмі я реалізував функцію-аналог вбудованої функції EVAL, яка працює для базових функцій мови LISP (cdr, car, cond, etc.) на мові LISP.

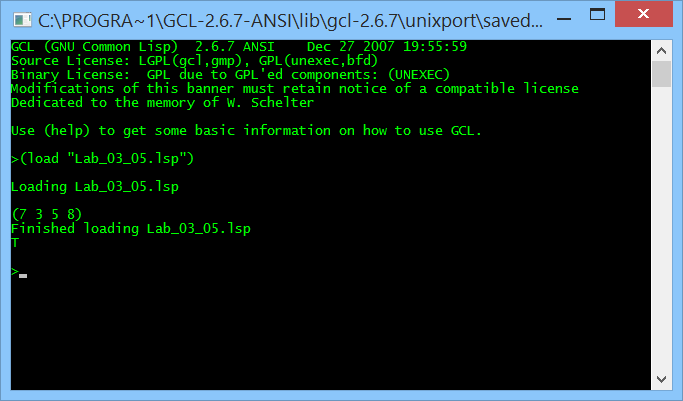
***Завдання 5***

|  |  |
| --- | --- |
| *Варіант* | *Задача* |
| 6 | Функція об’єднання двох списків. |

Код програми:

|  |
| --- |
| (defun my\_eval (loc\_form &optional (loc\_links nil))  (cond  ((atom loc\_form)  (cond  ((eq loc\_form 't1) 't1)  ((eq loc\_form 'nil) 'nil1)  ((numberp loc\_form) loc\_form)  ((car (assoc loc\_form loc\_links)))  (t (format t  "~%In atom absent local link: ~S"  loc\_form))  ((atom (car loc\_form))  (cond  ((eq (car loc\_form) 'quote1)  (cadr loc\_form))  ((eq (car loc\_form) 'cond1)  (eval-cond (cdr loc\_form) loc\_links))  ((get (car loc\_form) 'fn)  (my\_apply (get (car loc\_form) 'fn)  (eval-list (cdr loc\_form)  loc\_links)  loc\_links))  (t (my\_apply (car loc\_form)  (eval-list (cdr loc\_form)  loc\_links)  loc\_links))))  (t (my\_apply (car loc\_form)  (eval-list (cdr loc\_form) loc\_links)  loc\_links)))))  )    (defun eval-cond (branches context)  (cond  ((null branches) 'nil1)  ((not (eq (my\_eval (caar branches) context)  'nil1))  (my\_eval (cadar branches) context))  (t (eval-cond (cdr branches) context)))  )  (defun unite (w v)  (cond ((null w) v)  ((member (car w) v) (union (cdr w) v))  ((cons (car w) (union (cdr w) v))))  )  (defun my\_apply (func arg loc\_links)  (cond ((atom func)  (cond  ((eq func 'car1)  (cond ((eq (car arg) 'nil1)  'nil1)  (t (caar arg))))  ((eq func 'cdr1)  (cond ((eq (car arg) 'nil1)  'nil1)  ((null (cdar arg))  'nil1)  (t (cdar arg))))  ((eq func 'cons1)  (cond ((eq (cadr arg)  'nil1)  (list (car arg)))  (t (cons (car arg)  (cadr arg)))))  ((eq func 'atom1)  (cond ((atom (car arg))  't1)  (t 'nil1)))  ((eq func 'equal1)  (cond ((equal (car arg)  (cadr arg))  't1)  (t 'nil1)))  (t (format t "~%Unkown function:  ~S" func))))  ((eq (car func) 'lambda1)  (my\_eval (caddr func)  (create-links (cadr func)  arg loc\_links)))  ((eq (car func) 'unite)  (unite (caddr func)))  (t (format t  "~%It's not lambda call: ~S"  func)))  )  (defun create-links  (loc\_forms params env)  (cond  ((null loc\_forms) env)  (t (acons (car loc\_forms)  (car params)  (create-links (cdr loc\_forms)  (cdr params)  env))))  )  (defun eval-list (params loc\_links)  (cond  ((null params) nill)  (t (cons  (my\_eval (car params) loc\_links)  (eval-list (cdr params)  loc\_links))))  )  (print (my\_eval ‘(unite (3 5 7) (3 5 8)))) |

Скріншот виконання програми:



Висновок:

В даній програмі я реалізував функцію-аналог вбудованої функції EVAL, яка працює для базових функцій мови LISP (cdr, car, cond, etc.) на мові LISP. Також я добавив реалізацію функції яка об’єднує два списка в один.

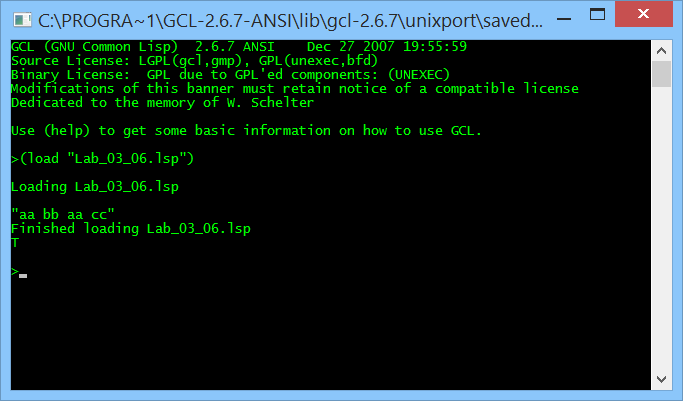
***Завдання 6***

|  |  |
| --- | --- |
| *Варіант* | *Задача* |
| 6 | Дано текст. Замінити в кожному реченні кожне входження заданого слова на задане нове слово. |

Код програми:

|  |
| --- |
| (defun substitute-word (z a s)  (format nil "~{~a~^ ~}"  (substitute  (read-from-string z)  (read-from-string a)  (read-from-string  (concatenate 'string "(" s ")"))))) |

Скріншот виконання програми:



Висновок:

В даній програмі я реалізував фунцію заміни входження одних шаблонів в строці на інші задані на мові LISP.

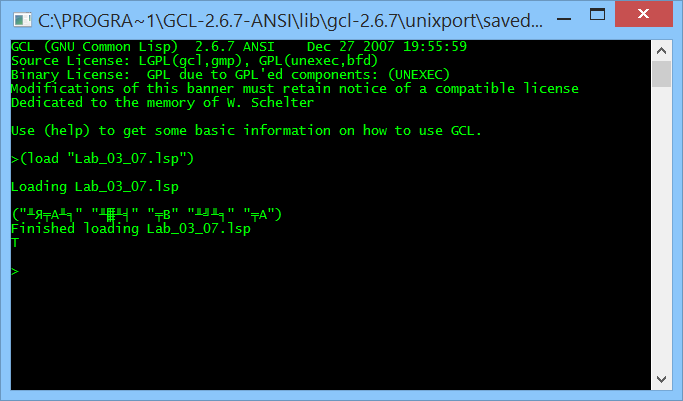
***Завдання 7***

|  |  |
| --- | --- |
| *Варіант* | *Задача* |
| 6 | Дана фраза українською (російською) мовою. Написати програму, що розбиває кожне слово фрази на склади. |

Код програми:

|  |
| --- |
| (setq vowels '(а е ё и о у ы э ю я))  (defun splitStr (src pat /)  (setq wordlist (list))  (setq len (strlen pat))  (setq cnt 0)  (setq letter cnt)  (while (setq cnt (vl-string-search pat src letter))  (setq word (substr src (1+ letter) (- cnt letter)))  (setq letter (+ cnt len))  (setq wordlist (append wordlist (list word)))  )  (setq wordlist (append wordlist (list (substr src (1+ letter)))))  )    (defun is\_vowels(chr lst\_vowels)  (member chr lst\_vowels))    (defun ins (s)  (cond ((is\_vowels s vowels) (pack (list s '-)))  (t s)  )  )    (defun divide\_word (word)  (cond  ((null word) nil)  (cons (ins (car word)) (divide\_word (cdr word)))  )  )    (defun syllables (txt)  (mapcar #'(lambda (s) (pack (divide\_word (unpack s)))) txt)  )    (defun divide\_string (txt)  (mapcar #'(lambda (s) (syllables s)) (splitStr txt))  )  (print (divide\_string "Привет мир")) |

Скріншот виконання програми:



Висновок:

В даній програмі я реалізував фунцію розбиття слів в строці на склади на мові LISP.

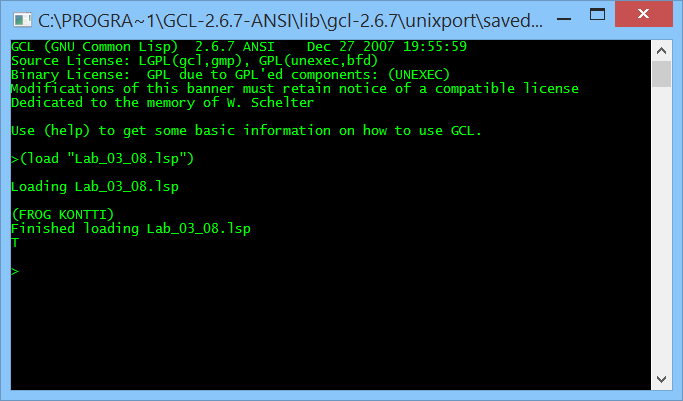
***Завдання 8***

|  |  |
| --- | --- |
| *Варіант* | *Задача* |
| 6 | «Мова пліткаря». Написати програму перекладу речення деякої мови на задану «таємну» мову. |

Код програми:

|  |
| --- |
| ; (deli 'kontti) -> ((#\K #\O) (#\N #\T #\T #\I))  (defun deli (word)  (deli-slovo nil  (coerce (string word) 'list)))  (defun deli-slovo (begin end)  (cond  ((null end) (list begin end))  ((sogl? (first end))  (deli-slovo  (v-end begin (first end))  (rest end)))  ((dolgaya-nach? end)  (list (append begin  (list (first end)  (second end)))  (cddr end)))  (t (list (v-end begin (first end))  (rest end)))))  (defun v-end (spisok element)  (append spisok (list element)))  (defun glasnaya? (letter)  (member letter \*glas\*))  (setq \*glas\* '(#\A #\E #\I #\O #\U #\Y #\a #\o));  (defun sogl? (letter)  (not (glasnaya? letter)))  (defun dolgaya-nach? (word)  (and (glasnaya? (first word))  (eql (first word)  (second word))))  (defun perevedi-slovo(word key)  (let ((chastislova (deli word))  (chastikey (deli key)))  (dolgota-glasnoi (first chastislova)  (second chastislova)  (first chastikey)  (second chastikey))))  (defun dolgota-glasnoi (begin1 end1 begin2 end2)  (cond  ((dolgaya-kon? begin1)  (cond  ((dolgaya-kon? begin2)  (pom-chasti begin1 end1 begin2 end2))  (t (pom-chasti (ukoroti begin1) end1  (udlinni begin2) end2))))  ((dolgaya-kon? begin2)  (pom-chasti  (udlinni begin1) end2  (ukoroti begin2) end2))  (t (pom-chasti begin1 end1 begin2 end2))))  (defun dolgaya-kon? (word)  (dolgaya-nach? (reverse word)))  (defun ukoroti (slog)  (if (not (rest slog))  nil  (cons (first slog)  (ukoroti (rest slog)))))  (defun udlinni (slog)  (if (null (rest slog))  (cons (first slog) slog)  (cons (first slog)  (udlinni (rest slog)))))  (defun pom-chasti  (begin1 end1 begin2 end2)  (list (sozv begin1 end1)  (sozv begin2 end2)))  (defun sozv (begin end)  (cond (;(perednee begin);  (soedeni begin (vpered end)))  (t (soedeni begin (nazad end)))))  (defun vpered (word)  (sublis  '((#\U . #\Y) (#\A . #\a) (#\O . #\o))  word))  (defun nazad (word)  (sublis  '((#\Y . #\U) (#\a . #\A) (#\o . #\O))  word))  (defun soedeni(begin end)  (intern (coerce (append begin end)  'string)))    (print (perevedi-slovo 'frog 'kontti)) |

Скріншот виконання програми:



Висновок:

В даній програмі я реалізував фунцію сплітника за заданим алгоритмом на мові LISP.

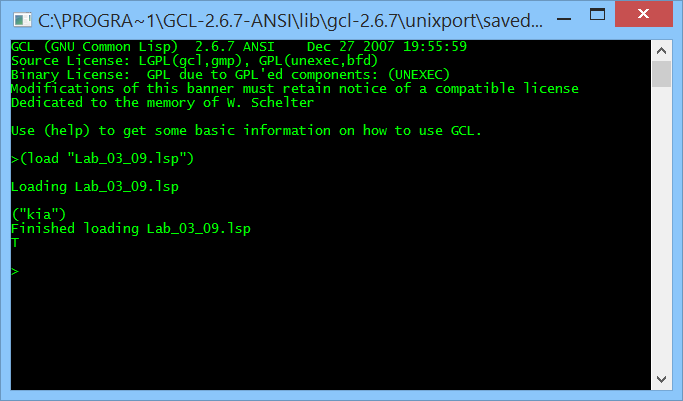
***Завдання 9***

|  |  |
| --- | --- |
| *Варіант* | *Задача* |
| 6 | Написати програму, яка виключає в початковому тексті з кожного слова його закінчення за словником. Словник закінчень представляти списком рядків. |

Код програми:

|  |
| --- |
| (defun string-right (s n)  (subseq s (- (length s) n)))    (defun drop-suffix (s v &aux (m (length s)) (n (length (car v))))  (cond ((null v) s)  ((string= (string-right s n) (car v)) (subseq s 0 (- m n)))  ((drop-suffix s (cdr v)))))    (defun drop-suffixes (w v &aux (v> (sort v #'string>)))  (mapcar #'(lambda (s) (drop-suffix s v>)) w))    (print (drop-suffixes '("kiamoto") '("moto" "iamoto"))) |

Скріншот виконання програми:



Висновок:

В даній програмі я реалізував функцію, яка виключає в початковому тексті з кожного слова його закінчення за словником на мові LISP.

**Висновок**: у ході виконання лабораторної роботи я розглянув різні види локальних визначень та особливості їх використання в рекурсивних програмах практично. Ознайомився з застосуванням техніки низхідної і висхідної рекурсії при написанні рекурсивних функцій з використанням локальних визначень. Порівняв можливості локальних визначень LET і LAMBDA по організації обчислень в рекурсивних програмах.