

Міністерство освіти та науки України  
Національний технічний університет України  
“Київський політехнічний інститут”  
Кафедра АСОІУ

ЗВІТ  
про виконання комп'ютерного практикуму №3  
з дисципліни  
“Мультипарадигменне програмування”  
Тема: Локальные определения и функционалы

Перевірів:  
Баклан І.В.

Виконав:  
студент 3-го курсу  
групи ПІ-52  
Набоков Е.М.

Київ 2017

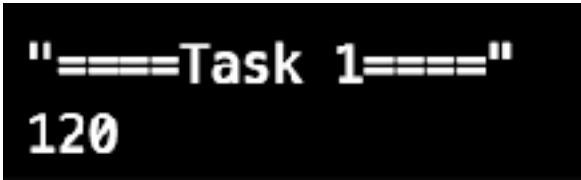
## Завдання 1

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

### *Програмный код*

```
(SETQ L1 '(1 (2 2 3) 4 (3 2 3) 5))  
(SETQ L2 '(3 2 3 2))  
  
;; task 1  
(DEFUN factorial (number)  
  (COND ((EQUAL number 0) 1)  
        (T  
         (LAMBDA (x y)(* x y)  
           number (factorial (- number 1))  
         )  
        )  
  )  
)  
  
(print "====Task 1====")  
(print (factorial 5))
```

### *Результат програми*



```
"====Task 1===="  
120
```

## 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.

### *Программный код*

```
(DEFUN derivative (f x)
  (COND ((numberp f) 0)
        ((EQUAL f x) 1)
        ((EQUAL (CAR f) '+)
         `(+ ,(derivative (CADR f) x) ,(derivative (CADDR f) x)))
        ((EQUAL (CAR f) '-')
         `(- ,(derivative (CADR f) x) ,(derivative (CADDR f) x)))
        ((EQUAL (CAR f) '* )
         `(+ (* ,(CADDR f) ,(derivative (CADR f) x)) (* ,(CADR f) ,(derivative (CADDR f) x))))
        ((EQUAL (CAR f) 'sin)
         `(* (cos ,(CADR f)) ,(derivative (CADR f) x)))
        ((EQUAL (CAR f) 'cos)
         `(* (- (cos ,(CADR f)) ,(derivative (CADR f) x))))
        ((EQUAL (CAR f) 'exp)
         `(* (exp ,(CADR f)) (derivative (CADR f) x)))
        ((EQUAL (CAR f) '^)
         `(* (* ,(CADDR f) (^ ,(CADR f) (- ,(CADDR f) 1))) ,(derivative (CADR f) x))))
```

### *Результат програми*

```
"===Task 2==="
(+ (* Y (+ 1 0)) (* (+ Y 3) 1))
```

### Завдання 3

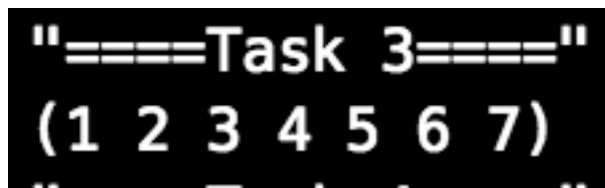
Решить задачу из лабораторной работы 2 с применением lambda, let

*Програмний код*

```
(DEFUN mymerge (l1 l2)
  (let ((res (make-array (+ (length l1)
                           (length l2))
                        :fill-pointer 0)))
    (loop for idx from 0 to (+ (length l1) (length l2)) do
      (let ((x (car l1))
            (y (car l2)))
        (when (and (not (null x)) (not (null y)))
          (if (<= x y)
              (progn
                (setf l1 (cdr l1))
                (vector-push x res))
              (progn
                (setf l2 (cdr l2))
                (vector-push y res))))))
    (mapcar #'(lambda (e) (vector-push e res)) (append l1 l2))
    (coerce res 'list)))

(print "====Task 3====")
(print (mymerge '(5 6 7) '(1 2 3 4)))
```

*Результат програми*



```
"====Task 3===="
(1 2 3 4 5 6 7)
"====Task 3===="
```

## Завдання 4

Реализовать программу - простейший интерпретатор Лисовских программ. На вход подается текст который может быть интерпретирован как вызов или суперпозиция функций листа, пример '(cons(car(cdr '(e r t w))) (cons (cdr '(gh 6)) nil)). Программа должна обеспечивать выполнение такого рода примеров.

Требования к программе:

- Должна обеспечивать интерпретацию базовых функций листа и арифметических операций
- В программе должны использоваться локальные определения
- Не допускается использование встроенной функции-интерпретатора EVAL;

*Программный код*

```
(DEFUN evaluate (local_form_conn &optional (local_connection nil))
  (COND
    ((atom local_form_conn)
      (COND
        ((EQUAL local_form_conn 't1) 't1)
        ((EQUAL local_form_conn 'nil) 'nil1)
        ((numberp local_form_conn) local_form_conn)
        ((CAR (ASSOC local_form_conn local_connection))))
      (t (format t
        "~%In atom absent local link: ~S"
        local_form_conn))
      ((atom (CAR local_form_conn))
        (COND
          ((EQUAL (CAR local_form_conn) 'quote1)
            (CADR local_form_conn))
          ((EQUAL (CAR local_form_conn) 'COND1)
            (condition_evaluate (CDR local_form_conn) local_connection))
          ((get (CAR local_form_conn) 'fn)
            (applyToExpression (get (CAR local_form_conn) 'fn)
              (listEvaluation (CDR local_form_conn)
                local_connection)
                local_connection))
          (t (applyToExpression (CAR local_form_conn)
            (listEvaluation (CDR local_form_conn)
              local_connection)
              local_connection))))
      (t (applyToExpression (CAR local_form_conn)
        (listEvaluation (CDR local_form_conn) local_connection)
        local_connection))))
  )

(DEFUN condition_evaluate (branches context)
  (COND
    ((NULL branches) 'nil1)
    ((not (EQUAL (evaluate (CAAR branches) context)
```

```

        'nil1))
      (evaluate (CADAR branches) context))
    (t (condition_evaluate (CDR branches) context)))
  )

(DEFUN applyToExpression (FUNC arg local_connection)
  (COND ((atom FUNC)
    (COND
      ((EQUAL FUNC 'CAR1)
        (COND ((EQUAL (CAR arg) 'nil1)
          'nil1)
          (t (CAAR arg))))
      ((EQUAL FUNC 'CDR1)
        (COND ((EQUAL (CAR arg) 'nil1)
          'nil1)
          ((NULL (CDAR arg))
            'nil1)
          (t (CDAR arg))))
      ((EQUAL FUNC 'cons1)
        (COND ((EQUAL (CADR arg)
          'nil1)
          (list (CAR arg))
          (t (cons (CAR arg)
            (CADR arg)))))
      ((EQUAL FUNC 'atom1)
        (COND ((atom (CAR arg))
          't1)
          (t 'nil1)))
      ((EQUAL FUNC 'myEqualLua_)
        (COND ((myEqualLua (CAR arg)
          (CADR arg))
          't1)
          (t 'nil1)))
        (t (format t "~%Unkown FUNCtion:
          ~S" FUNC))))
    ((EQUAL (CAR FUNC) 'lambda1)
      (evaluate (CADDR FUNC)
        (connectionHandler (CADR FUNC)
          arg local_connection)))
    (t (format t
      "~%It's not lambda call: ~S"
      FUNC)))
  )

(DEFUN connectionHandler
  (local_form_conns params env)
  (COND

```

```

((NULL local_form_conns) env)
(t (ACONS (CAR local_form_conns)
          (CAR params)
          (connectionHandler (CDR local_form_conns)
                             (CDR params)
                             env))))
)

(DEFUN listEvaluation (params local_connection)
  (COND
    ((NULL params) NIL)
    (t (cons
         (evaluate (CAR params) local_connection)
         (listEvaluation (CDR params)
                        local_connection))))
)

(print "====Task 4====")
(print (evaluate (+ (* 5 3) 2)))

```

*Результат програми*

```

"====Task 4===="
17

```

## Завдання 5

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

*Програмний код*

```
(DEFUN evaluate (local_form_conn &optional (local_connection nil))
  (COND
    ((atom local_form_conn)
      (COND
        ((EQUAL local_form_conn 't1) 't1)
        ((EQUAL local_form_conn 'nil) 'nil1)
        ((numberp local_form_conn) local_form_conn)
        ((CAR (ASSOC local_form_conn local_connection)))
        (t (format t
          "~%In atom absent local link: ~S"
          local_form_conn))
      ))
    ((atom (CAR local_form_conn))
      (COND
        ((EQUAL (CAR local_form_conn) 'quote1)
          (CADR local_form_conn))
        ((EQUAL (CAR local_form_conn) 'COND1)
          (condition_evaluate (CDR local_form_conn) local_connection))
        ((get (CAR local_form_conn) 'fn)
          (applyToExpression (get (CAR local_form_conn) 'fn)
            (listEvaluation (CDR local_form_conn)
              local_connection)
            local_connection))
        (t (applyToExpression (CAR local_form_conn)
          (listEvaluation (CDR local_form_conn)
            local_connection)
            local_connection))))
    (t (applyToExpression (CAR local_form_conn)
      (listEvaluation (CDR local_form_conn) local_connection)
      local_connection))))
)

(DEFUN condition_evaluate (branches context)
  (COND
    ((NULL branches) 'nil1)
    ((not (EQUAL (evaluate (CAAR branches) context)
      'nil1))
      (evaluate (CADAR branches) context))
    (t (condition_evaluate (CDR branches) context)))
)

(DEFUN intersect (A B)
  (if (EQUAL A ())
```



A

```
(if (member (CAR A) B)
    (cons (CAR A) (intersect (CDR A) B))
    (intersect (CDR A) B))))
```

```
(DEFUN applyToExpression (func arg local_connection)
  (COND ((atom func)
    (COND
      ((EQUAL func 'CAR1)
        (COND ((EQUAL (CAR arg) 'nil1)
          'nil1)
          (t (CAAR arg))))
      ((EQUAL func 'CDR1)
        (COND ((EQUAL (CAR arg) 'nil1)
          'nil1)
          ((NULL (CDAR arg))
            'nil1)
          (t (CDAR arg))))
      ((EQUAL func 'cons1)
        (COND ((EQUAL (CADR arg)
          'nil1)
          (list (CAR arg))
          (t (cons (CAR arg)
            (CADR arg)))))
      ((EQUAL func 'atom1)
        (COND ((atom (CAR arg))
          't1)
          (t 'nil1)))
      ((EQUAL func 'myEqualLua)
        (COND ((myEqualLua_ (CAR arg)
          (CADR arg))
          't1)
          (t 'nil1)))
        (t (format t "~%Unkown function:
          ~S" func))))
    ((EQUAL (CAR func) 'lambda1)
      (evaluate (CADDR func)
        (connectionHandler (CADR func)
          arg local_connection)))

    ((EQUAL (CAR func) 'intersect)
      (intersect (CADDR func)))

    (t (format t
      "~%It's not lambda call: ~S"
      func)))
  )
```

```

(DEFUN connectionHandler
  (local_form_conns params env)
  (COND
    ((NULL local_form_conns) env)
    (t (ACONS (CAR local_form_conns)
              (CAR params)
              (connectionHandler (CDR local_form_conns)
                                (CDR params)
                                env))))
  )

(DEFUN listEvaluation (params local_connection)
  (COND
    ((NULL params) NIL)
    (t (cons
        (evaluate (CAR params) local_connection)
        (listEvaluation (CDR params)
                        local_connection))))
  )

(print "====Task 5====")
(print (evaluate (intersect '(3 5 7) '(3 5 8))))

```

9.	Функция пересечения множеств.
----	-------------------------------

*Результат програми*

```

"====Task 5===="
(3 5)

```

## Завдання 7

Дана фраза українського (росського) мови. Написати програму, яка розбиває кожне слово на слоги

*Программний код*

:: Task 7

```
(SETQ vowels '(а е ё и о у ы э ю я))
```

```
(DEFUN split (src pat /)
  (SETQ splittedWords (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 splittedWords (append splittedWords (list word))))
  )
  (SETQ splittedWords (append splittedWords (list (substr src (1+ letter))))))
)
```

```
(DEFUN is_vowels(CHR lst_vowels)
  (member CHR lst_vowels))
```

```
(DEFUN check_in (s)
  (COND ((is_vowels s vowels)
    (pack (list s '-)))
    (t s)
  )
)
```

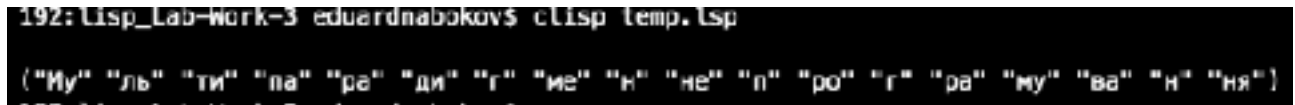
```
(DEFUN divWords (word)
  (COND
    ((NULL word) NIL)
    (CONS (check_in (car word)) (divWords (CDR word)))
  )
)
```

```
(DEFUN content (txt)
  (MAPCAR #'(lambda (s)
    (pack (divWords (unpack s)))) txt)
)
```

```
(DEFUN divStrs (txt)
  (MAPCAR '(lambda (s) (content s))
```

```
(split txt))  
)  
  
(print (divStrs "Мультипарадигменне програмування"))
```

*Результат програми*



```
192: Cisp_Lab-Work-3 eduardnabokov$ cCisp temp.lsp  
( 'Му' 'ль' 'ти' 'па' 'ра' 'ди' 'г' 'ме' 'н' 'не' 'п' 'ро' 'г' 'ра' 'му' 'ва' 'н' 'ня' )
```

## Завдання 8

Єсть ключове слово, наприклад, “сплетня”. Слово переводиться на мову сплетника шляхом відокремлення першого слога в переводимому і ключовому слові (наприклад, сло - во і спле - тня) з наступною перестановкою за певними правилами : „(слово спле тня) перетворюється в „(сплево слотня). Кожне слово перетворюється в пару слів. Перше слово є конкатенація першого слога ключового слова і частини переводимого слова, залишеної після відокремлення від нього першого слога. Друге слово є конкатенація першого слога переводимого слова і частини ключового слова, залишеної після відокремлення від нього першого слога.

*Программний код*

```
;; Task 8  
(DEFUN dividing (word)  
  (divWord nil  
    (coerce (string word) 'list)))  
  
(DEFUN divWord (begin end)  
  (COND  
    ((null end)  
     (list begin end))  
    ((letter?  
     (first end))  
     (divWord  
       (toTheEnd begin (first end))  
       (left end)))  
    ((long? end)  
     (list (append begin  
                   (list (first end)  
                       (nextOneS end)))  
           (CDDR end)))  
    (T  
     (list (toTheEnd begin (first end))  
           (left end)))))
```

```
(DEFUN toTheEnd (ll element)
  (append ll (list element)))
```

```
(DEFUN vowel? (letter)
  (member letter *vow1*))
(SETQ *vow1*
  '(#\A #\E #\I #\O #\U #\Y #\a #\o));
(DEFUN letter? (letter)
  (NOT (vowel? letter)))
```

```
(DEFUN long? (word)
  (AND (vowel? (first word))
    (EQUAL (first word)
      (nextOneS word)))))
```

```
(DEFUN translateWord(word key)
  (LET ((partsOfWord (dividing word))
    (partsOfKey (dividing key)))
    (len_of_vowels (first partsOfWord)
      (nextOneS partsOfWord)
      (first partsOfKey)
      (nextOneS partsOfKey)))))
```

```
(DEFUN len_of_vowels (begin_first end1 begin_second end2)
  (COND
    ((long_? begin_first)
      (COND
        ((long_? begin_second)
          (parts begin_first end1 begin_second end2))
        (T (parts (shortened begin_first) end1
          (enlong begin_second) end2))))
    ((long_? begin_second)
      (parts
        (enlong begin_first) end2
        (shortened begin_second) end2))
    (T
      (parts begin_first end1 begin_second end2)))))
```

```
(DEFUN long_? (word)
  (long? (reverse word)))
```

```
(DEFUN shortened (partWord)
  (IF
    (NOT (left partWord))
    nil
    (CONS (first partWord)
      (shortened (left partWord)))))
```

```
(DEFUN enlong (partWord)
  (IF (null (left partWord))
```

```

(CONS (first partWord) partWord)
(CONS (first partWord)
(enlong (left partWord))))))

(DEFUN parts
  (begin_first end1 begin_second end2)
  (list (conn begin_first end1)
  (conn begin_second end2)))

(DEFUN conn (begin end)
  (COND ((before begin)
  (merge_s begin (forward end)))
  (t (merge_s begin (backward end)))))

(DEFUN forward (word)
  (subPart
'((#\U . #\Y) (#\A . #\a) (#\O . #\o))
word))

(DEFUN backward (word)
  (subPart
'((#\Y . #\U) (#\a . #\A) (#\o . #\O))
word))
(DEFUN merge_s(begin end)
(intern (coerce (append begin end)
'string)))

(print (translateWord 'issue))

```

*Результат програми*

## **Завдання 9**

Написать программу исключающую в исходном тексте из каждого слова его окончание по словарю Словарь окончаний представлять списком строк.

*Программный код*

```

(defun sideString(s n)
  (subseq s (- (length s) n)))

(defun removePart (s v &aux (m (length s)) (n (length (car v))))
  (cond ((null v) s)
        ((string= (sideString s n) (car v)) (subseq s 0 (- m n)))
        ((removePart s (cdr v)))))

```

```
(defun removePartes (w v &aux (v> (sort v #'string>)))  
  (mapcar #'(lambda (s) (removePart s v>)) w))
```

```
(print (removePartes '("sunnyday") '("day" "moon")))
```

*Результат програми*

```
192:lisp_Lab-Work-3 eduardnabokov$ clisp task8.lsp  
("sunny")
```