Міністерство освіти і науки України

Національний технічний університет України „КПІ”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки

інформації та управління

**ЗВІТ**

до лабораторної роботи № 3

з предмету:

„Мультипарадигменне програмування”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Виконав**  **студент** |  | *ІП-61 Кушка Михайло Олександрович, 3-й курс, ІП-6116* |  |  |
|  |  | (№ групи, прізвище, ім’я, по батькові, курс, номер залікової книжки) |  |  |
|  |  |  |  |  |
| **Прийняв** |  | *Очеретяний О. К.* |  |  |
|  |  | (посада, прізвище, ім’я, по батькові ) |  |  |

Київ 2018

ЗМІСТ

[1. постановка задачі 3](#_Toc526712203)

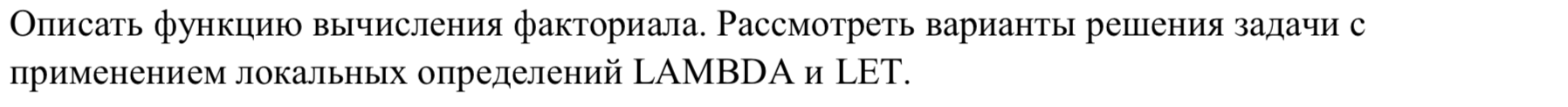
[2. Результати роботи 5](#_Toc526712204)

[3. Висновок 6](#_Toc526712205)

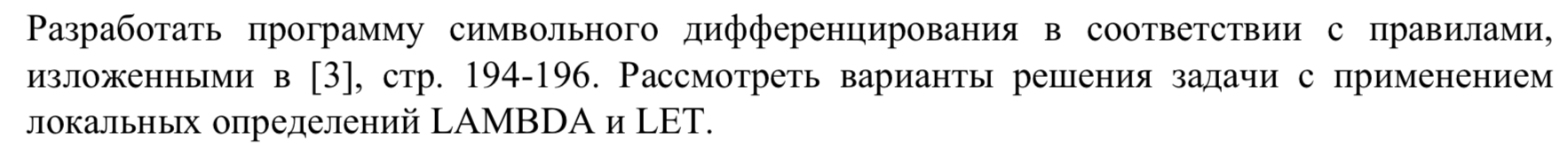
[4. Код програми 7](#_Toc526712206)

# постановка задачі

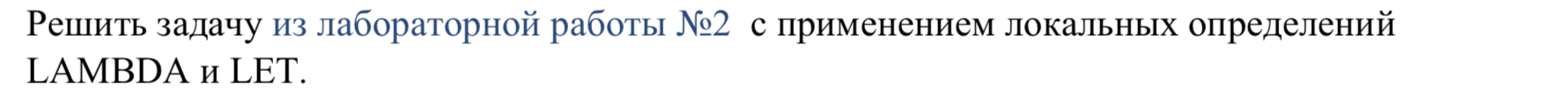
**Задание 1**

****

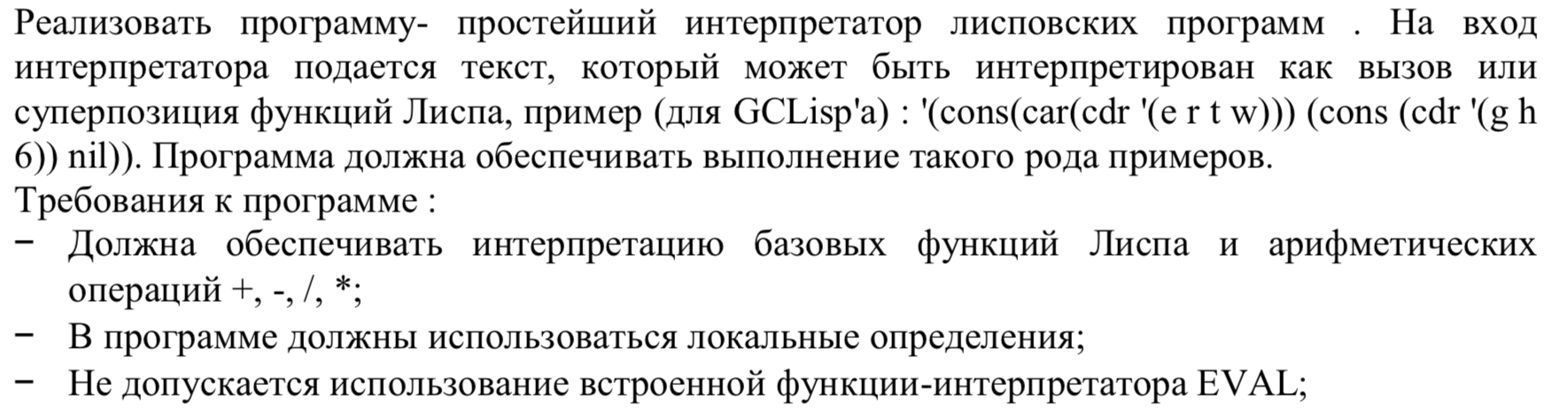
**Задание 2**



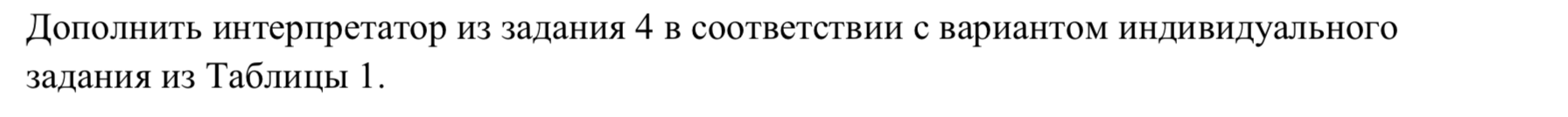
**Задание 3**

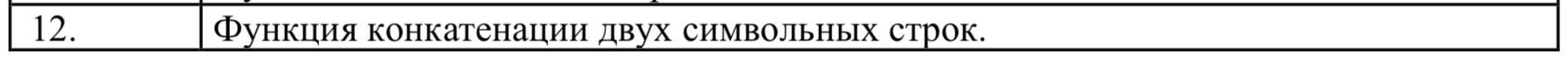


**Задание 4**

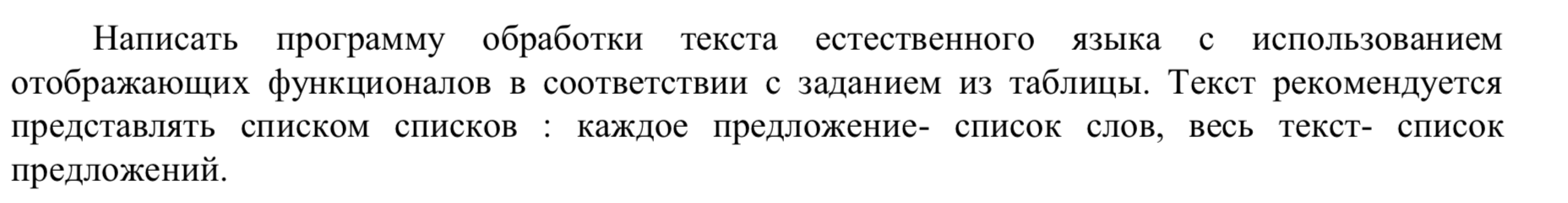


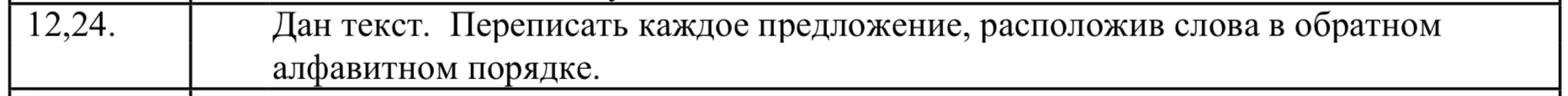
**Задание 5**



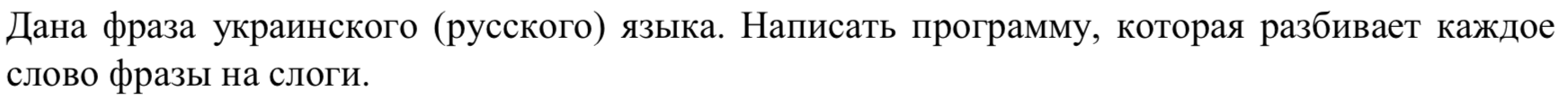


**Задание 6**

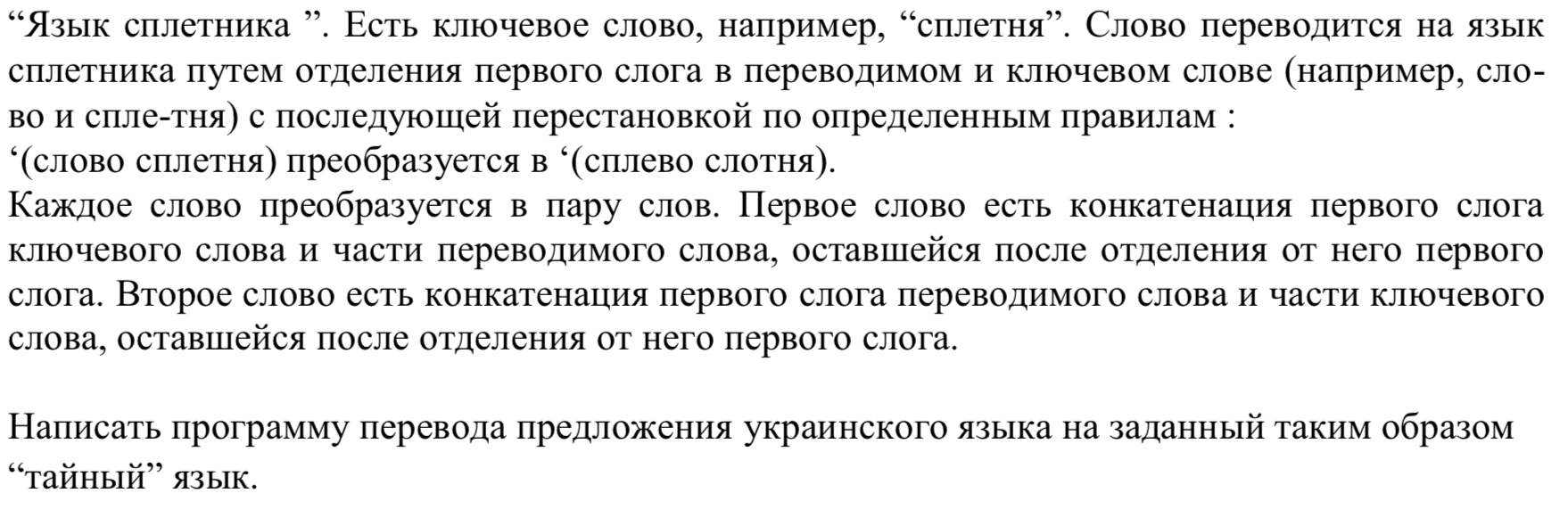




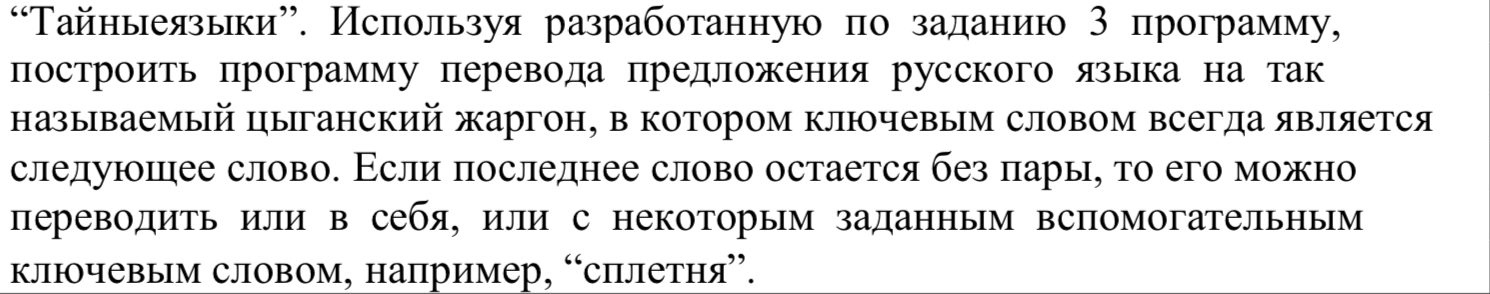
**Задание 7**



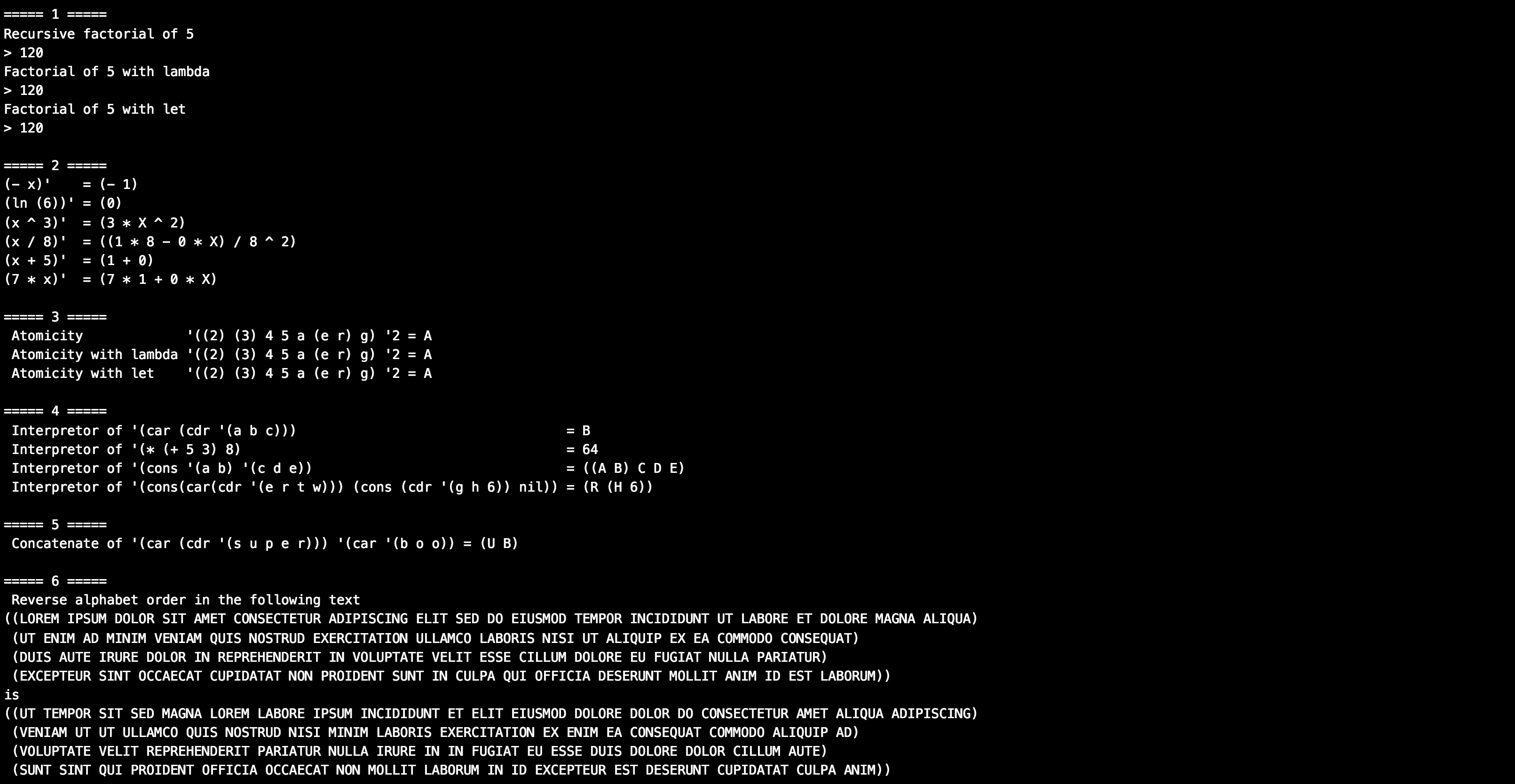
**Задание 8**

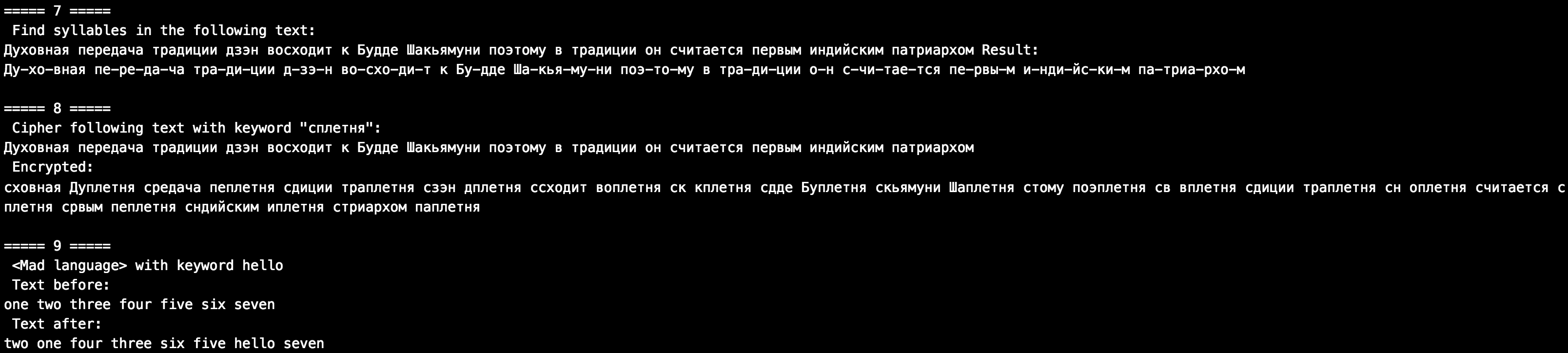


**Задание 9**



# Результати роботи





# Висновок

Особливим у даній лабораторній роботі була необхідність проектування програми до її безпосередньої реалізації. Це пов’язано з тим, що дані завдання не були шаблонними, як і простими. Складнощі виникали скоріше не з реалізацією алгоритму, а з чіткою і правильною його розробкою.

# Код програми

;;;;;;;;;;;

;;;; 1 ;;;;

;;;;;;;;;;;

; Recursive function

(defun factorial (n)

(cond

((eq n 1) 1)

(t (\* n (factorial (- n 1))))))

; Lambda recursive function

(defun factorial1 (n)

((lambda (n)

(cond

((eq n 1) 1)

(t (\* n (factorial1 (- n 1))))))

n)

)

; Let recursive function

(defun factorial2 (n)

(let ((n

(cond

((eq n 1) 1)

(t (\* n (factorial2 (- n 1))))

)

))

n)

)

(format t "~%===== 1 =====")

(format t "~%Recursive factorial of 5~%> ~D" (factorial 5))

(format t "~%Factorial of 5 with lambda~%> ~D" (factorial1 5))

(format t "~%Factorial of 5 with let~%> ~D" (factorial2 5))

;;;;;;;;;;;

;;;; 2 ;;;;

;;;;;;;;;;;

(defun atom-diff (expr)

(cond

((null expr) nil)

((atom expr)

(cond

((eq expr 'x) 1)

(t 0)))

((not (atom expr))

(cond

; ln(x)

((and (eq (first expr) 'ln) (eq (car (second expr)) 'x))

(list '1 '/ 'x))

; ln(c)

((eq (first expr) 'ln)

(list 0))

((eq (first expr) '-)

(list '- (atom-diff (second expr))))

(t nil)) )

(t nil)))

(defun is-number (n)

(cond

((and (atom n) (member n '(0 1 2 3 4 5 6 7 8 9))) t)

(t nil)))

(defun pair-diff (expr)

(let

((a (first expr))

(b (third expr))

(sign (second expr)))

(cond

; + or -

((or (eq sign '+) (eq sign '-))

(list (atom-diff a) sign (atom-diff b)))

; \*

((eq sign '\*)

(list a '\* (atom-diff b) '+ (atom-diff a) '\* b))

; /

((eq sign '/)

(list (list (atom-diff a) '\* b '- (atom-diff b) '\* a ) '/ b '^ '2))

; ^

((eq sign '^)

(cond

((and (eq a 'x) (is-number b))

(list b '\* a '^ (- b 1)))

(t nil)))

(t t))))

(format t "~%~%===== 2 =====")

(format t "~%(- x)' = ~D" (atom-diff '(- x)))

(format t "~%(ln (6))' = ~D" (atom-diff '(ln (6))))

(format t "~%(x ^ 3)' = ~D" (pair-diff '(x ^ 3)))

(format t "~%(x / 8)' = ~D" (pair-diff '(x / 8)))

(format t "~%(x + 5)' = ~D" (pair-diff '(x + 5)))

(format t "~%(7 \* x)' = ~D" (pair-diff '(7 \* x)))

;;;;;;;;;;;

;;;; 3 ;;;;

;;;;;;;;;;;

; Recursive function

(defun atomicity (lst n)

(cond ((null lst) nil)

((and (atom (car lst)) (eq n '0)) (car lst))

((not (atom (car lst)))

(atomicity (cdr lst) n))

((and (atom (car lst)) (not (eq n '0)))

(atomicity (cdr lst) (- n 1)))))

; Lambda recursive function

(defun atomicity1 (lst n)

((lambda (lst n)

(cond ((null lst) nil)

((and (atom (car lst)) (eq n '0)) (car lst))

((not (atom (car lst)))

(atomicity1 (cdr lst) n))

((and (atom (car lst)) (not (eq n '0)))

(atomicity1 (cdr lst) (- n 1)))))

lst n)

)

; Let recursive function

(defun atomicity2 (lst n)

(let ((lst

(cond ((null lst) nil)

((and (atom (car lst)) (eq n '0)) (car lst))

((not (atom (car lst)))

(atomicity2 (cdr lst) n))

((and (atom (car lst)) (not (eq n '0)))

(atomicity2 (cdr lst) (- n 1)))))

)

lst)

)

(format t "~%~%===== 3 =====")

(format t "~% Atomicity '((2) (3) 4 5 a (e r) g) '2 = ~D" (atomicity '((2) (3) 4 5 a (e r) g) '2))

(format t "~% Atomicity with lambda '((2) (3) 4 5 a (e r) g) '2 = ~D" (atomicity1 '((2) (3) 4 5 a (e r) g) '2))

(format t "~% Atomicity with let '((2) (3) 4 5 a (e r) g) '2 = ~D" (atomicity2 '((2) (3) 4 5 a (e r) g) '2))

;;;;;;;;;;;

;;;; 4 ;;;;

;;;;;;;;;;;

(defun interpretor (expr)

(cond

((null expr) nil)

((atom expr) expr)

((eq (car expr) 'car) (car (interpretor (second expr))))

((eq (car expr) 'cdr) (cdr (interpretor (second expr))))

((eq (car expr) '\*) (\* (interpretor (second expr)) (interpretor (third expr))))

((eq (car expr) '/) (/ (interpretor (second expr)) (interpretor (third expr))))

((eq (car expr) '+) (+ (interpretor (second expr)) (interpretor (third expr))))

((eq (car expr) '-) (- (interpretor (second expr)) (interpretor (third expr))))

((eq (car expr) 'cons) (cons (interpretor (second expr)) (interpretor (third expr))))

(t (second expr))))

(format t "~%~%===== 4 =====")

(format t "~% Interpretor of '(car (cdr '(a b c))) = ~D" (interpretor '(car (cdr '(a b c)))))

(format t "~% Interpretor of '(\* (+ 5 3) 8) = ~D" (interpretor '(\* (+ 5 3) 8)))

(format t "~% Interpretor of '(cons '(a b) '(c d e)) = ~D" (interpretor '(cons '(a b) '(c d e))))

(format t "~% Interpretor of '(cons(car(cdr '(e r t w))) (cons (cdr '(g h 6)) nil)) = ~D" (interpretor '(cons(car(cdr '(e r t w))) (cons (cdr '(g h 6)) nil))))

;;;;;;;;;;;

;;;; 5 ;;;;

;;;;;;;;;;;

(defun concat (a b)

(cond

((null a) b)

((null b) a)

(t (list (interpretor a) (interpretor b)))))

(format t "~%~%===== 5 =====")

(format t "~% Concatenate of '(car (cdr '(s u p e r))) '(car '(b o o)) = ~D" (concat '(car (cdr '(s u p e r))) '(car '(b o o))))

;;;;;;;;;;;

;;;; 6 ;;;;

;;;;;;;;;;;

; QuickSort

(defun quicksort (l)

(cond

((null l) nil)

(T

(append

(quicksort (list>= (car l) (cdr l)))

(list (car l))

(quicksort (list< (car l) (cdr l)))))))

(defun list< (a b)

(cond

((or (null a) (null b)) nil)

((string< a (car b)) (list< a (cdr b)))

(T (cons (car b) (list< a (cdr b))))))

(defun list>= (a b)

(cond

((or (null a) (null b)) nil)

((string>= a (car b)) (list>= a (cdr b)))

(T (cons (car b) (list>= a (cdr b))))))

(defun sentence-reverse (list)

(cond

((null list) nil)

(t (quicksort list))))

(defun text-reverse (text)

(cond

((null text) nil)

(t (cons (sentence-reverse (car text)) (text-reverse (cdr text))))))

(format t "~%~%===== 6 =====")

(format t "~% Reverse alphabet order in the following text ~%~D ~%is ~%~D" '(

(Lorem ipsum dolor sit amet consectetur adipiscing elit sed do eiusmod tempor incididunt ut labore et dolore magna aliqua)

(Ut enim ad minim veniam quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat)

(Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur)

(Excepteur sint occaecat cupidatat non proident sunt in culpa qui officia deserunt mollit anim id est laborum))

(text-reverse '(

(Lorem ipsum dolor sit amet consectetur adipiscing elit sed do eiusmod tempor incididunt ut labore et dolore magna aliqua)

(Ut enim ad minim veniam quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat)

(Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur)

(Excepteur sint occaecat cupidatat non proident sunt in culpa qui officia deserunt mollit anim id est laborum))))

;;;;;;;;;;;

;;;; 7 ;;;;

;;;;;;;;;;;

; https://habr.com/post/37448/

(defun word-split-syllables (word power)

(let

((vovels '(а о и е ё э ы у ю я)) ; 4

(sonorant '(м н л р)) ; 3

(voiced-consonants '(г ж б в д з й)) ; 2

(voicelessness '(к х п ф т с ш щ ч ц)) ; 1

(other '(ъ ь)))

(cond

((string= word "") nil)

((is-in-list (string-upcase (first-letter word)) vovels)

(concatenate 'string (first-letter word) (word-split-syllables (cut-head word) 4)))

((is-in-list (string-upcase (first-letter word)) sonorant)

(if (> 3 power)

(concatenate 'string (first-letter word) (word-split-syllables (cut-head word) 3))

(concatenate 'string "-" (first-letter word) (word-split-syllables (cut-head word) 0))))

((is-in-list (string-upcase (first-letter word)) voiced-consonants)

(if (> 2 power)

(concatenate 'string (first-letter word) (word-split-syllables (cut-head word) 2))

(concatenate 'string "-" (first-letter word) (word-split-syllables (cut-head word) 0))))

((is-in-list (string-upcase (first-letter word)) voicelessness)

(if (> 1 power)

(concatenate 'string (first-letter word) (word-split-syllables (cut-head word) 1))

(concatenate 'string "-" (first-letter word) (word-split-syllables (cut-head word) 0))))

((is-in-list (string-upcase (first-letter word)) other)

(concatenate 'string (first-letter word) (word-split-syllables (cut-head word) power)))

((is-in-list (string-upcase (first-letter word)) '(\_))

(concatenate 'string (first-letter word) (word-split-syllables (cut-head word) 0)))

(t ":(") ) ))

(defun first-letter (word)

(subseq word 0 1))

(defun cut-head (word)

(subseq word 1 (length word)))

(defun is-in-list (symbol list)

(cond

((null list) nil)

((string= symbol (write-to-string (car list))) t)

(t (is-in-list symbol (cdr list)))))

(defun replace-with (text from to)

(cond

((string= text "") nil)

((string= (first-letter text) from)

(concatenate 'string to (replace-with (cut-head text) from to)))

(t (concatenate 'string (first-letter text) (replace-with (cut-head text) from to)))))

(format t "~%~%===== 7 =====")

(format t "~% Find syllables in the following text: ~%~D Result: ~%~D"

"Духовная передача традиции дзэн восходит к Будде Шакьямуни поэтому в традиции он считается первым индийским патриархом"

((lambda (text)

(replace-with (word-split-syllables (replace-with text " " "\_") 0) "\_" " ")

) "Духовная передача традиции дзэн восходит к Будде Шакьямуни поэтому в традиции он считается первым индийским патриархом"))

;;;;;;;;;;;

;;;; 8 ;;;;

;;;;;;;;;;;

(defun get-word-part (word part)

(let

((w (word-split-syllables word 0)))

((lambda (pos text)

(cond

((not (search "-" w)) w)

((string= part "head")

(subseq text 0 pos))

((string= part "tail")

(replace-with (subseq text pos (length text)) "-" ""))

(t part) ))

(search "-" w) w)))

(defun cipher-word (word key)

(concatenate 'string

(get-word-part key "head") (get-word-part word "tail") " "

(get-word-part word "head") (get-word-part key "tail")))

(defun cipher-text (text key)

(cond

((string= text "") nil)

((not (search " " text))

(cipher-word text key))

(t

((lambda (pos)

(concatenate 'string

(cipher-word (subseq text 0 pos) key)

" "

(cipher-text (subseq text (+ pos 1) (length text)) key)))

(search " " text)))))

(format t "~%~%===== 8 =====")

(format t "~% Cipher following text with keyword \"сплетня\": ~%~D ~% Encrypted: ~%~D"

"Духовная передача традиции дзэн восходит к Будде Шакьямуни поэтому в традиции он считается первым индийским патриархом"

(cipher-text "Духовная передача традиции дзэн восходит к Будде Шакьямуни поэтому в традиции он считается первым индийским патриархом" "сплетня"))

;;;;;;;;;;;

;;;; 9 ;;;;

;;;;;;;;;;;

(defun swap (text pair)

(cond

((string= text "") nil)

((not (search " " text)) text)

((eq pair 0)

(concatenate 'string (swap (subseq text (+ (search " " text) 1) (length text)) 1) " " (subseq text 0 (search " " text))))

((eq pair 1)

(concatenate 'string (subseq text 0 (search " " text)) " " (swap (subseq text (+ (search " " text) 1) (length text)) 0)))

(t ":(")))

(defun index-of-second-space (text counter is-second)

(cond

((string= text "") nil)

((not (search " " text)) nil)

((not is-second)

(index-of-second-space (subseq text (+ (search " " text) 1) (length text)) (search " " text) t))

(is-second

(+ counter (search " " text) 1))

(t nil) ))

(defun mad-lang (text key)

(cond

((string= text "") nil)

((not (index-of-second-space text 0 nil))

(concatenate 'string key " " text))

(t

(let ((pos (index-of-second-space text 0 nil)))

(concatenate 'string (swap (subseq text 0 pos) 0) " " (mad-lang (subseq text (+ pos 1) (length text)) key)) )) ))

(format t "~%~%===== 9 =====")

(format t "~% <Mad language> with keyword ~D ~% Text before: ~%~D ~% Text after: ~%~D"

"hello"

"one two three four five six seven"

(mad-lang "one two three four five six seven" "hello"))