|  |
| --- |
| **МИНОБРНАУКИ РОССИИ**  федеральное государственное бюджетное образовательное  учреждение высшего образования  **«Национальный исследовательский университет «МЭИ»** |

|  |  |
| --- | --- |
| **Институт** | ИВТИ |
| **Кафедра** | ПМИИ |

|  |  |
| --- | --- |
|  |  |
| **Структуры данных и методы программирования**  **Отчет по Курсовой работе**  **Тема: «Программирование на языках Лисп, ФРЛ и Пролог»**  **Вариант 3**  **Выполнил: студент группы А-13а-19**  **Рамазанов Н. М.**  **Преподаватель: Чернов П. Л.**  **Москва, 2021** | |

**Часть 1 – Лисп.**

1. Реализовать функции **(@FINDLIST *object list test*)** и **(@FINDLIST-IF *test list*)**, осуществляющие поиск на верхнем уровне в списке list. В качестве результата возвращается список элементов из list, для которых (test object element) не NIL. Здесь element - текущий элемент списка list. Если аргумент test опущен, то test = EQUAL. Для второй функции предикат test является одноместным.

*(@FINDLIST obj list test)*

Код:

(defun @FINDLIST (obj lst test)

(cond ((not (listp lst)) 'non-list)

((null lst) 'list-empty)

(T (FINDLIST1 obj lst test)) ) )

(defun FINDLIST1 (obj lst test)

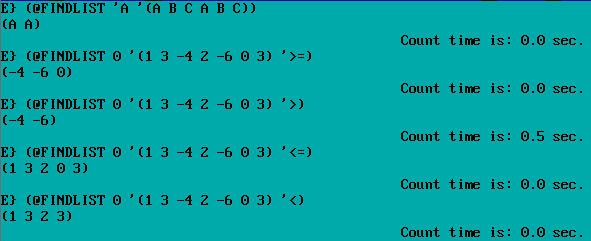
(if (null test) (setq test 'EQUAL) )

(cond ((null lst) nil)

((funcall test obj (car lst)) (cons (car lst) (FINDLIST1 obj (cdr lst) test)))

(T (FINDLIST1 obj (cdr lst) test) ) ) )

Тесты:



*(@FINDLIST-IF test list)*

Код:

(defun @FINDLIST-IF (test lst)

(cond

((not (listp lst)) 'non-list)

((null lst) 'list-empty)

(T (FINDLIST-IF1 test lst)) ) )

(defun FINDLIST-IF1 (test lst)

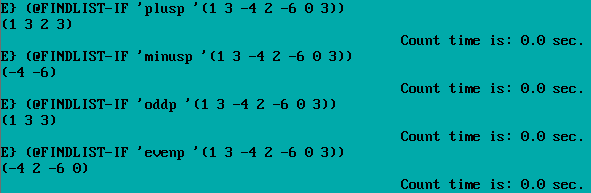
(cond

((null lst) nil)

((funcall test (car lst)) (cons (car lst) (FINDLIST-IF1 test (cdr lst))))

(T (FINDLIST-IF1 test (cdr lst))) ) )

Тесты:



2. Реализовать функцию **(@INSEND *atom list*)**, возвращающую список list, в котором в конец каждого подсписка добавлен атом atom.

а) рекурсивная реализация

Код:

(defun @INSEND-R (atm lst)

(cond

((null lst) nil)

((atom lst) nil)

(T (INSEND1-R atm lst)) ) )

(defun INSEND1-R (atm lst)

(cond

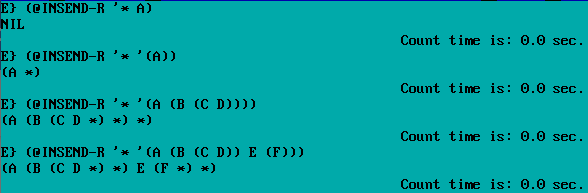
((null lst) (list atm))

((atom (car lst)) (cons (car lst) (INSEND1-R atm (cdr lst)) ))

((listp (car lst)) (cons (INSEND1-R atm (car lst)) (INSEND1-R atm (cdr lst)) ))

(T nil) ) )

Тесты:



б) итерационная реализация

Код:

(defun @INSEND-I (atm lst r)

(setq r nil)

(INSEND1-I

(cons atm

(loop

((null lst) r)

(push (cond

((atom (car lst)) (pop lst))

(T (@INSEND-I atm (pop lst))) ) r) ) )) )

(defun INSEND1-I (lst r)

(setq r nil)

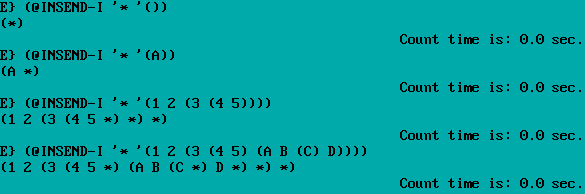
(loop

((null lst) r)

(push (car lst) r)

(pop lst) ) )

Тесты:



в) реализация с использованием функционала

Код:

(defun @INSEND-F (atm lst)

(cond

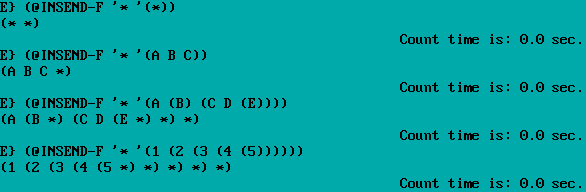
((atom lst) lst)

((listp lst) (append

(mapcar '(lambda (lst1)(@INSEND-F atm lst1)) lst)

(list atm)) ) ) )

Тесты:



**Часть 2 – ФРЛ.**

3. Реализовать функцию **(ANALYSIS *sentence*)**, позволяющую распознать синтаксически правильные предложения русского языка. Под синтаксически правильными предложениями будем понимать те, в которых правильный порядок слов и все слова предложения имеются в словаре.

Проверить работу функции на примере словаря, содержащего 12 – 15 слов. Использовать грамматику простых повествовательных предложений, содержащих подлежащее, сказуемое и, возможно, несколько определений.

*(ANALYSIS sentence)*

Код:

(defun ANALYSIS (sentence)

(setq s\_copy sentence)

(setq fl 0)

(setq struct (fget GRAMMAR))

(setq verb 0)

(loop

((or (null s\_copy) (= fl 1) ))

(setq fl1 (is\_in\_map (car s\_copy) map))

(cond

((null fl1)

(setq fl 1)

(princ "Word ") (princ (car s\_copy)) (princ " is not included in map! ")

null

)

(T

(setq pstruct (fget GRAMMAR (car struct)))

(cond

((equal (car (fget fl1 ЧАСТЬ-РЕЧИ)) (car pstruct))

(setq s\_copy (cdr s\_copy))

(cond

((null (cdr pstruct))

(setq struct (cdr struct))

)

(T

(setq pstruct (cdr pstruct))

)

)

(if (equal (car (fget fl1 ЧАСТЬ-РЕЧИ)) ГЛАГОЛ) (setq verb 1))

)

(T

(cond

((equal (car pstruct) ПРИЛАГАТЕЛЬНОЕ)

(setq struct (cdr struct))

)

(T (setq fl 1) )

)

)

)

)

)

)

(cond

((and (= fl 0) (= verb 1)) True)

(T False)

)

)

(deframeq GRAMMAR

(S1 ($value (ПРИЛАГАТЕЛЬНОЕ) (ПРИЛАГАТЕЛЬНОЕ)))

(S2 ($value (СУЩЕСТВИТЕЛЬНОЕ)))

(S3 ($value (ГЛАГОЛ)))

)

(deframeq big

(ЧАСТЬ-РЕЧИ ($value (ПРИЛАГАТЕЛЬНОЕ))) )

(deframeq comfortable

(ЧАСТЬ-РЕЧИ ($value (ПРИЛАГАТЕЛЬНОЕ))) )

(deframeq beautiful

(ЧАСТЬ-РЕЧИ ($value (ПРИЛАГАТЕЛЬНОЕ))) )

(deframeq clear

(ЧАСТЬ-РЕЧИ ($value (ПРИЛАГАТЕЛЬНОЕ))) )

(deframeq expensive

(ЧАСТЬ-РЕЧИ ($value (ПРИЛАГАТЕЛЬНОЕ))) )

(deframeq table

(ЧАСТЬ-РЕЧИ ($value (СУЩЕСТВИТЕЛЬНОЕ))) )

(deframeq chair

(ЧАСТЬ-РЕЧИ ($value (СУЩЕСТВИТЕЛЬНОЕ))) )

(deframeq teacup

(ЧАСТЬ-РЕЧИ ($value (СУЩЕСТВИТЕЛЬНОЕ))) )

(deframeq notebook

(ЧАСТЬ-РЕЧИ ($value (СУЩЕСТВИТЕЛЬНОЕ))) )

(deframeq stands

(ЧАСТЬ-РЕЧИ ($value (ГЛАГОЛ))) )

(deframeq lies

(ЧАСТЬ-РЕЧИ ($value (ГЛАГОЛ))) )

(deframeq works

(ЧАСТЬ-РЕЧИ ($value (ГЛАГОЛ))) )

(deframeq exists

(ЧАСТЬ-РЕЧИ ($value (ГЛАГОЛ))) )

(setq map (list big comfortable beautiful clear expensive table chair teacup notebook stands lies works exists))

(defun is\_in\_map (word lst)

(if (not (null lst))

(cond

((equal (car lst) word) (car lst))

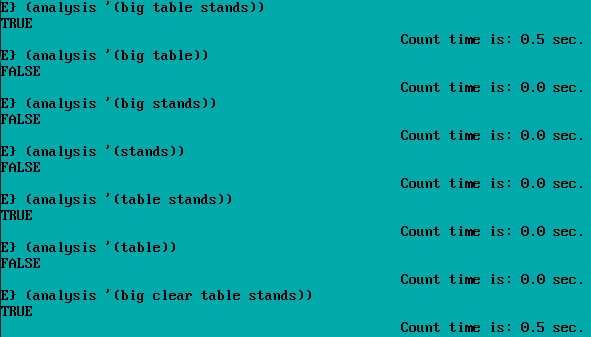
(T (is\_in\_map word (cdr lst)))

)

)

)

Тесты:



**Часть 3 – Пролог.**

1. Реализовать функции **(@FINDLIST *object list test*)** и **(@FINDLIST-IF *test list*)**, осуществляющие поиск на верхнем уровне в списке list. В качестве результата возвращается список элементов из list, для которых (test object element) не NIL. Здесь element - текущий элемент списка list. Если аргумент test опущен, то test = EQUAL. Для второй функции предикат test является одноместным.

Код:

domains

list = integer\*

predicates

findlist(integer, list, string, list)

findlist\_if(string, list, list)

clauses

findlist(OBJ, [H|[]], TEST, R1):- TEST="=", H=OBJ, R1=[H].

findlist(OBJ, [H|[]], TEST, R1):- TEST="=", H<>OBJ, R1=[].

findlist(OBJ, [H|T], TEST, R):- TEST="=", H=OBJ, findlist(OBJ, T, TEST, RES), R=[H|RES], !.

findlist(OBJ, [H|T], TEST, R):- TEST="=", H<>OBJ, findlist(OBJ, T, TEST, RES), R=RES, !.

findlist(OBJ, [H|[]], TEST, R1):- TEST="<", H<OBJ, R1=[H].

findlist(OBJ, [H|[]], TEST, R1):- TEST="<", H>=OBJ, R1=[].

findlist(OBJ, [H|T], TEST, R):- TEST="<", H<OBJ, findlist(OBJ, T, TEST, RES), R=[H|RES], !.

findlist(OBJ, [H|T], TEST, R):- TEST="<", H>=OBJ, findlist(OBJ, T, TEST, RES), R=RES, !.

findlist(OBJ, [H|[]], TEST, R1):- TEST=">", H>OBJ, R1=[H].

findlist(OBJ, [H|[]], TEST, R1):- TEST=">", H<=OBJ, R1=[].

findlist(OBJ, [H|T], TEST, R):- TEST=">", H>OBJ, findlist(OBJ, T, TEST, RES), R=[H|RES], !.

findlist(OBJ, [H|T], TEST, R):- TEST=">", H<=OBJ, findlist(OBJ, T, TEST, RES), R=RES, !.

findlist(OBJ, [H|[]], TEST, R1):- TEST="<=", H<=OBJ, R1=[H].

findlist(OBJ, [H|[]], TEST, R1):- TEST="<=", H>OBJ, R1=[].

findlist(OBJ, [H|T], TEST, R):- TEST="<=", H<=OBJ, findlist(OBJ, T, TEST, RES), R=[H|RES], !.

findlist(OBJ, [H|T], TEST, R):- TEST="<=", H>OBJ, findlist(OBJ, T, TEST, RES), R=RES, !.

findlist(OBJ, [H|[]], TEST, R1):- TEST=">=", H>=OBJ, R1=[H].

findlist(OBJ, [H|[]], TEST, R1):- TEST=">=", H<OBJ, R1=[].

findlist(OBJ, [H|T], TEST, R):- TEST=">=", H>=OBJ, findlist(OBJ, T, TEST, RES), R=[H|RES], !.

findlist(OBJ, [H|T], TEST, R):- TEST=">=", H<OBJ, findlist(OBJ, T, TEST, RES), R=RES, !.

findlist(OBJ, [H|[]], TEST, R1):- TEST="<>", H<>OBJ, R1=[H].

findlist(OBJ, [H|[]], TEST, R1):- TEST="<>", H=OBJ, R1=[].

findlist(OBJ, [H|T], TEST, R):- TEST="<>", H<>OBJ, findlist(OBJ, T, TEST, RES), R=[H|RES], !.

findlist(OBJ, [H|T], TEST, R):- TEST="<>", H=OBJ, findlist(OBJ, T, TEST, RES), R=RES, !.

findlist\_if(TEST, [H|[]], R1):- TEST="+", H>0, R1=[H].

findlist\_if(TEST, [H|[]], R1):- TEST="+", H<=0, R1=[].

findlist\_if(TEST, [H|T], R):- TEST="+", H>0, findlist\_if(TEST, T, RES), R=[H|RES], !.

findlist\_if(TEST, [H|T], R):- TEST="+", H<=0, findlist\_if(TEST, T, RES), R=RES, !.

findlist\_if(TEST, [H|[]], R1):- TEST="-", H<0, R1=[H].

findlist\_if(TEST, [H|[]], R1):- TEST="-", H>=0, R1=[].

findlist\_if(TEST, [H|T], R):- TEST="-", H<0, findlist\_if(TEST, T, RES), R=[H|RES], !.

findlist\_if(TEST, [H|T], R):- TEST="-", H>=0, findlist\_if(TEST, T, RES), R=RES, !.

findlist\_if(TEST, [H|[]], R1):- TEST="odd", H mod 2 = 1, R1=[H].

findlist\_if(TEST, [H|[]], R1):- TEST="odd", H mod 2 = 0, R1=[].

findlist\_if(TEST, [H|T], R):- TEST="odd", H mod 2 = 1, findlist\_if(TEST, T, RES), R=[H|RES], !.

findlist\_if(TEST, [H|T], R):- TEST="odd", H mod 2 = 0, findlist\_if(TEST, T, RES), R=RES, !.

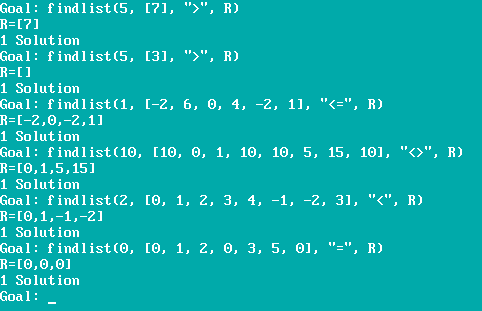
findlist\_if(TEST, [H|[]], R1):- TEST="even", H mod 2 = 0, R1=[H].

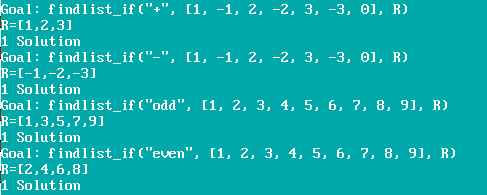
findlist\_if(TEST, [H|[]], R1):- TEST="even", H mod 2 = 1, R1=[].

findlist\_if(TEST, [H|T], R):- TEST="even", H mod 2 = 0, findlist\_if(TEST, T, RES), R=[H|RES], !.

findlist\_if(TEST, [H|T], R):- TEST="even", H mod 2 = 1, findlist\_if(TEST, T, RES), R=RES, !.

Тесты:





3. Реализовать функцию **(ANALYSIS *sentence*)**, позволяющую распознать синтаксически правильные предложения русского языка. Под синтаксически правильными предложениями будем понимать те, в которых правильный порядок слов и все слова предложения имеются в словаре.

Проверить работу функции на примере словаря, содержащего 12 – 15 слов. Использовать грамматику простых повествовательных предложений, содержащих подлежащее, сказуемое и, возможно, несколько определений.

Код:

domains

sentence = string\*

database

noun(string)

verb(string)

adj(string)

predicates

analysis(sentence)

sent(sentence, sentence)

noun\_group(sentence, sentence)

verb\_group(sentence, sentence)

clauses

analysis(L):- sent(L, L1),!,analysis(L1).

analysis([]).

sent(L, L0):- noun\_group(L, L1), verb\_group(L1, L0).

noun\_group([X|L], L):- noun(X), !.

noun\_group([X,Y|L], L):- adj(X), noun(Y), !.

noun\_group([X,Y,Z|L], L):- adj(X), adj(Y), noun(Z), !.

verb\_group([X|L], L):- verb(X),!.

noun("table"). noun("chair"). noun("teacup"). noun("notebook").

adj("big"). adj("comfortable"). adj("beautiful"). adj("clear"). adj("expensive").

verb("stands"). verb("lies"). verb("works"). verb("exists").

Тесты:

