Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных технологий

Кафедра математики и цифровых технологий

**ОТЧЕТ**

по лабораторной работе № 5

по дисциплине «Функционально-логическое программирование»

ОГУ 02.03.02. 4025. 583 О

Руководитель

Старший преподаватель кафедры МЦТ,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И.В. Минина

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

Студент группы 22ФИИТ(б)РАИС

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И.Д. Евдокимов

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

Оренбург 2025

**1) вычислить длину списка, т.е. количество элементов в списке;**

**Код**

% Реализация, для которой вложенный список - один элемент.

list\_len1([],0).

list\_len1([\_|T],Len):- list\_len1(T,Len1), Len is 1 + Len1.

% Реализация, которая подсчитывает фактическое количество атомарных элементов

list\_len2([],0).

list\_len2([H|T],Len):- atomic(H), not(H=[]), list\_len2(T,Len1), Len is 1 + Len1.

list\_len2([H|T],Len):- atomic(H), H=[], list\_len2(T,Len1), Len is Len1.

list\_len2([H|T],Len):- not(atomic(H)), list\_len2(H,Len1), list\_len2(T,Len2), Len is Len1 + Len2.

% Реализация через встроенный предикат

list\_len3(List, Res):- length(List, Res).

**Запросы:**

?- list\_len1([],Res).

Res = 0 ;

?- list\_len1([1],Res).

Res = 1 ;

?- list\_len1([1,2,3],Res).

Res = 3 ;

?- list\_len1([1,[2],3],Res).

Res = 3 ;

?- list\_len1([[],[],[]],Res).

Res = 3 ;

?- list\_len1([[1,2,3],[1,2,3],[1,2,3]],Res).

Res = 3 ;

\*\*

?- list\_len2([],Res).

Res = 0 ;

?- list\_len2([1],Res).

Res = 1 ;

?- list\_len2([1,2,3],Res).

Res = 3 ;

?- list\_len2([1,[2],3],Res).

Res = 3 ;

?- list\_len2([[],[],[]],Res).

Res = 0 ;

?- list\_len2([[1,2,3],[1,2,3],[1,2,3]],Res).

Res = 9 ;

\*\*

?- list\_len3([],Res).

Res = 0 ;

?- list\_len3([1],Res).

Res = 1 ;

?- list\_len3([1,2,3],Res).

Res = 3 ;

?- list\_len3([1,[2],3],Res).

Res = 3 ;

?- list\_len3([[],[],[]],Res).

Res = 3 ;

?- list\_len3([[1,2,3],[1,2,3],[1,2,3]],Res).

Res = 3 ;

**2) проверить принадлежность элемента списку. Предикат будет иметь два аргумента: первый — искомое значение, второй — список, в котором производится поиск;**

**Код:**

% реализация для поиска элементов на одном уровне (без учёта вложенности)

is\_member1(\_, []):- fail.

is\_member1(Value, [H|T]):- H=Value ; is\_member1(Value,T).

% реализация для поиска элементов на всех уровнях (учитывает вложенность)

is\_member2(\_, []):- fail.

is\_member2(Value, [H|T]):- atomic(H), (H=Value ; is\_member2(Value, T)).

is\_member2(Value, [H|T]):- not(atomic(H)), (H=Value ; is\_member2(Value, H) ; is\_member2(Value, T)).

% Реализация через встроенный предикат

is\_member3(Value, List):- member(Value, List).

**Запросы:**

?- is\_member1(1, []).

false.

?- is\_member1(1, [2,3,4]).

false.

?- is\_member1(1, [1,2,3,4]).

true.

?- is\_member1(1, [[1]]).

false.

?- is\_member1(1, [[1],2,3]).

false.

?- is\_member1(1, [[[1]],2,3]).

false.

?- is\_member1(1, [[[[1]]],2,3]).

false.

?- is\_member1([1], []).

false.

?- is\_member1([1], [2,3,4]).

false.

?- is\_member1([1], [1,2,3,4]).

false.

?- is\_member1([1], [[1]]).

true.

?- is\_member1([1], [[1],2,3]).

true.

?- is\_member1([1], [[[1]],2,3]).

false.

\*\*

?- is\_member2(1, []).

false.

?- is\_member2(1, [2,3,4]).

false.

?- is\_member2(1, [1,2,3,4]).

true.

?- is\_member2(1, [[1]]).

true.

?- is\_member2(1, [[1],2,3]).

true.

?- is\_member2(1, [[[1]],2,3]).

true.

?- is\_member2(1, [[[[1]]],2,3]).

true.

?- is\_member2([1], []).

false.

?- is\_member2([1], [2,3,4]).

false.

?- is\_member2([1], [1,2,3,4]).

false.

?- is\_member2([1], [[1]]).

true.

?- is\_member2([1], [[1],2,3]).

true.

?- is\_member2([1], [[[1]],2,3]).

true.

\*\*

?- is\_member3(1, []).

false.

?- is\_member3(1, [2,3,4]).

false.

?- is\_member3(1, [1,2,3,4]).

true.

?- is\_member3(1, [[1]]).

false.

?- is\_member3(1, [[1],2,3]).

false.

?- is\_member3(1, [[[1]],2,3]).

false.

?- is\_member3(1, [[[[1]]],2,3]).

false.

?- is\_member3([1], []).

false.

?- is\_member3([1], [2,3,4]).

false.

?- is\_member3([1], [1,2,3,4]).

false.

?- is\_member3([1], [[1]]).

true.

?- is\_member3([1], [[1],2,3]).

true.

?- is\_member3([1], [[[1]],2,3]).

false.

**3) соединить два списка в один. Первые два аргумента предиката будут представлять соединяемые списки, а третий — результат соединения;**

**Код:**

% Реализация через встроенный предикат

concatanation1(List1, List2, Res):- append(List1, List2, Res).

% Реализация вручную

concatanation2([], List2, List2).

concatanation2([H|T], List2, [H|T1]):- concatanation2(T, List2, T1).

**Запросы:**

?- concatanation1([1], [], Res).

Res = [1] ;

?- concatanation1([1,2,3], [], Res).

Res = [1, 2, 3] ;

?- concatanation1([], [1], Res).

Res = [1] ;

?- concatanation1([], [1,2,3], Res).

Res = [1, 2, 3] ;

?- concatanation1([1,2],[3,4], Res).

Res = [1, 2, 3, 4] ;

?- concatanation1([4,3],[2,1], Res).

Res = [4, 3, 2, 1] ;

\*\*

?- concatanation2([1], [], Res).

Res = [1] ;

?- concatanation2([1,2,3], [], Res).

Res = [1, 2, 3] ;

?- concatanation2([], [1], Res).

Res = [1] ;

?- concatanation2([], [1,2,3], Res).

Res = [1, 2, 3] ;

?- concatanation2([1,2],[3,4], Res).

Res = [1, 2, 3, 4] ;

?- concatanation2([4,3],[2,1], Res).

Res = [4, 3, 2, 1] ;

**4) «обратить» список (записать его элементы в обратном порядке). Предикат будет иметь два аргумента: первый — исходный список, второй — список, получающийся в результате записи элементов первого аргумента в обратном порядке;**

**Код:**

% Реализация через встроенный предикат

reverse1(List, RList):- reverse(List, RList).

% Реализация вручную через аккумулирование

reverse2(List, Reversed) :- reverse2(List, [], Reversed).

reverse2([], Acc, Acc).

reverse2([H|T], Acc, Reversed) :- reverse2(T, [H|Acc], Reversed).

**Запросы:**

% ?- reverse1([], Res).

Res = [] ;

% ?- reverse1([1, 2, 3, 4, 5], Res).

Res = [5, 4, 3, 2, 1] ;

% ?- reverse1([1, 2, [3, 4, 5]], Res).

Res = [[3, 4, 5], 2, 1] ;

% ?- reverse1([1], Res).

Res = [1] ;

% ?- reverse1([1, 2, 1], Res).

Res = [1, 2, 1] ;

\*\*

% ?- reverse2([], Res).

Res = [] ;

% ?- reverse2([1, 2, 3, 4, 5], Res).

Res = [5, 4, 3, 2, 1] ;

% ?- reverse2([1, 2, [3, 4, 5]], Res).

Res = [[3, 4, 5], 2, 1] ;

% ?- reverse2([1], Res).

Res = [1] ;

% ?- reverse2([1, 2, 1], Res).

Res = [1, 2, 1] ;

**5) проверить, является ли список палиндромом. Палиндромом называется список, который совпадает со своим обращением. Соответственно, у данного предиката будет всего один аргумент (список, который проверяем на "палиндромность");**

**Код:**

% Реализация обращения

reverse2(List, Reversed) :- reverse2(List, [], Reversed).

reverse2([], Acc, Acc).

reverse2([H|T], Acc, Reversed) :- reverse2(T, [H|Acc], Reversed).

% Реализация проверки на палиндром

is\_palindrom(List):- reverse2(List, Reversed), Reversed=List.

**Запросы:**

?- is\_palindrom([]).

true.

?- is\_palindrom([1]).

true.

?- is\_palindrom([1, 2, 3]).

false.

?- is\_palindrom([1, 2, 1]).

true.

?- is\_palindrom([[1, 2, 3], a, [1, 2, 3]]).

true.

**6) получать элемент списка по его номеру так же, как по номеру можно получать элемент массива в императивных языках программирования. Предикат будет трехаргументный: первый аргумент — исходный список, второй аргумент — номер элемента и третий — элемент списка, указанного в качестве первого аргумента предиката, имеющий номер, указанный в качестве второго аргумента;**

**Код:**

get\_element([H|\_], 0, H).

get\_element([\_|T], Number, Element):-

not(Number=0),

Nextnumber is Number - 1,

get\_element(T, Nextnumber, Element).

**Запросы:**

% ?- get\_element([a, b, c, d, e], 0, Res).

Res = a ;

% ?- get\_element([a, b, c, d, e], 1, Res).

Res = b ;

% ?- get\_element([a, b, c, d, e], 2, Res).

Res = c ;

% ?- get\_element([a, b, c, d, e], 3, Res).

Res = d ;

% ?- get\_element([a, b, c, d, e], 4, Res).

Res = e ;

% ?- get\_element([a, b, c, d, e], 5, Res).

false.

**7) удалять все вхождения заданного значения из списка. Предикат будет зависеть от трех параметров. Первый параметр будет соответствовать удаляемому списку, второй — исходному значению, а третий — результату удаления из первого параметра всех вхождений второго параметра.**

**Код:**

deleter([], \_, Acc, Acc).

deleter([Element|T], Element, Acc, Outputlist):- deleter(T, Element, Acc, Outputlist).

deleter([Notelement|T], Element, Acc, Outputlist):- not(Notelement=Element), deleter(T, Element, [Notelement|Acc], Outputlist).

deleteall(Inputlist, Element, Outputlist):- deleter(Inputlist, Element, [], Outputlistreversed), reverse(Outputlistreversed, Outputlist).

**Запросы:**

?- deleteall([], 1, Res).

Res = [] ;

?- deleteall([1, 1, 1, 1, 1], 1, Res).

Res = [] ;

?- deleteall([[a], [a], a, [a]], a, Res).

Res = [[a], [a], [a]] ;

?- deleteall([[a], [a], a, [a]], [a], Res).

Res = [a] ;

?- deleteall([1, a, 2, a, 3, a], a, Res).

Res = [1, 2, 3] ;

?- deleteall([1, 3, 3, 3, 1], 1, Res).

Res = [3, 3, 3] ;