Zad1.

1. A) Proszę zdefiniować predykat member(X,L) ktory jest spelniony jezeli X jest elementem listy L.
2. member(X, [X|\_]). % member(X, [Head|Tail]) is true if X = Head
3. % that is, if X is the head of the list
4. member(X, [\_|Tail]) :- % or if X is a member of Tail,
5. member(X, Tail). % ie. if member(X, Tail) is true.

B) Proszę zdefiniować predykat member(X,L) ktory jest spelniony jezeli X jest elementem listy L na jakimkolwiek poziomie : We all know the classic Prolog predicate for member:

rmember(4, [1,2,[3,4,[5]],[6,7,9]]).

rmember(X, [X|\_]). %% (1)

rmember(X, [H|\_]) :- %% (2)

rmember(X, H).

rmember(X, [\_|T]) :- %% (3)

rmember(X, T).

2A)Proszę zdefiniować predykat suffix(L1, L2), jesli lista L1 jest koncem listy L2

prefix([],\_).

prefix([A|B],[A|C]) :- prefix(B,C).

suffix(A,B) :- reverse(A,AR), reverse(B,BR), prefix(AR,BR).

2B) palindrom([a,b,a]).

|  |
| --- |
| pal([]). |
|  |

|  |
| --- |
| pal(X) :- |
|  |

|  |
| --- |
| reverse(Rev, X), |
|  |

|  |
| --- |
| X=Rev. |
| pal([]).  pal([\_]).  pal(Pal) :-  append([H|T], [H], Pal),  pal(T). |

3. A) Zdefiniuj split(X,L,L1,L2) L1 wszyskie el mniejsze rowne X z listy L , a L2 wszyskie elementy wieksze od X

?- split(5,[2,7,4,8,-1,5],L1,L2).

L1 = [2,4,-1,5],

L2 = [7,8]

split2(L1,L1,L2,L2).

split(X,[],L1,L2,NL1,NL2):- split2(L1,NL1,L2,NL2).

split(X, L, L1, L2):- split(X, L, L1, L2, [], []).

split(X, [H0|T0], L1, L2, NL1, NL2):-

H0 @> X, !, append(NL1,[H0],NNL1),split(X, T0, L1, L2, NNL1, NL2);

append(NL2,[H0],NNL2),split(X, T0, L1, L2, NL1, NNL2).

4. Drzewo binarne można reprezentować przez term nil – puste drzewo , albo drzewo(X,L,P) z elementem Xi poddrzewami L i P

a) search(D,X), ktory jest spelniony jesli X jest elementem drzewa D

search(X,X).

search(d(X,L,R),N) :-

search(X,N);

search(L,N);

search(R,N).

Test: search(d(0,d(1,d(3,nil,nil),d(4,nil,nil)),d(2,d(5,nil,nil),d(6,nil,nil))),2)

b) prod(D,P), ktory jest spelniony jesli P jest iloczynem drzewa D

prod(nil, 1).

prod(d(X,\_,\_), X).

prod(d(X,L,R), N) :-

print(L),

prod(L, N1), prod(R, N2),

N is X\*N1\*N2,

print(N).

TEST: prod(d(1,d(1,d(3,nil,nil),d(4,nil,nil)),d(2,d(5,nil,nil),d(1,nil,nil))),120)

prod(nil,1). prod(t(V,nil,nil),V). prod(t(K,L,R),N) :- prod(L,W1), prod(R,W2), N is K\*W1\*W2

c) postorder(D,L) lista L zawiera D w kolejnosci postorder <http://users.utcluj.ro/~cameliav/lp/lab8.pdf>

postorder(tree(X, L, R), Xs):-

postorder(L, Ls),

postorder(R, Rs),

append(Ls, Rs, Xs1),

append(Xs1, [X], Xs).

postorder(void, []).



%------1A ----------------------------------------------

del2(\_,[],[]).

del2(X,[\_|L1],[\_|L2]):- del2(X,L1,L2).

del(X,[\_|L1],[\_|L2]):- del(X,L1,L2).

del(X,[X|L1],L2):-del2(X,L1,L2).

%------1B ----------------------------------------------

times(X,L1,Y):-times2(X,L1,Y,[]).

times2(X,[H1|L1],L2,R2):- R is H1\*X, append(R2,[R],R3),

times2(X,L1,L2,R3).

times2(\_,[],Y,R3):-times(R3,Y).

times(X,X).

%----------------------------- To Co zrobilismy żeby wypisywał

down2(\_,L,L).

down(X,L) :- down(X,L,[]).

down(0,L,N2):- down2(\_,L,N2).

down(X,L,N2):- append(N2,[X],R), X2 is X-1, down(X2,L,R).

%------2A--------------------- to co powinno być

downn(N,[N|L1]):- F is N-1, downn(F,L1).

downn(-1,[]).

%------2B ----------------------------------------------

neig(X,Y,[\_,H2|T1]):- neig(X,Y,[H2|T1]).

neig(X,Y,[X,Y|\_]).

neig(X,Y,[Y,X|\_]).

%------3 ----------------------------------------------

pos(X):- X>0.

neg(X):- X<0.

filler(\_,[],[]).

filler(\_,L,[]).

filler(Pred, [H1|L1], [H1|L2]):- call(Pred,H1), !, filler(Pred,L1,L2).

filler(Pred,[H1|L1],[H2|L2]):- filler(Pred,L1,[H2|L2]).

%------3 ----------------------------------------------

Search

Leaves

leaves(d(\_,nil,nil),1).

leaves(d(\_,L,P),N) :-

leaves(L,N1),

leaves(P,N2),

N is N1+N2.

Plus(N,D1,D2)

pluss(N,nil,nil).

pluss(N,d(X1,L1,P1), d(X2,L2,P2)) :-

pluss(N,L1,L2),

pluss(N,P1,P2),

X2 is N+X1.