%1 Calc:

f(X,Y,R) :- X>(-1), Y<1, R is (X+Y-2),!.

f(X,Y,R) :- R is X-Y.

g(X,Y,R) :- R is X\*X - Y\*Y.

e(X,Y,R) :- f(X,Y,R1), g(X,Y,R2), R is R1+3\*R2.

**%2 CMMDC / CMMMC**

cmmdc(X,Y,Y) :- mod(X,Y) =:= 0, !.

cmmdc(X,Y,D) :- Z is mod(X,Y), cmmdc(Y,Z,D).

cmmmc(X,Y,M) :- cmmdc(X,Y,D), M is X\*Y/D.

h(A,B,C,R) :- cmmdc(A,C,R1), cmmmc(A,B,R2), cmmdc(B,C,R3), R is R1-R2+R3.

cmmdc3(A,B,C,R) :- cmmdc(A,B,R1), cmmdc(R1,C,R).

maxc(A,B,C,M) :- cmmdc(A,C,R1), cmmmc(A,B,R2), cmmdc(B,C,R3), max(R1,R2,REZ1), max(REZ1,R3,M).

%Operatorul cut "!", următoarea linie va fii ca un "else"

maxim(A,B,A) :- A>=B, !.

maxim(A,B,B).

%Max din 3 nr

maxim3(A,B,C,MAX3) :- maxim(A,B,X), maxim(X,C,MAX3).

maxim4(A,B,C,MAX3) :- maxim(A,B,X), maxim(X,C,MAX3).

% f(X,Y) = X+Y-2, daca X>-1, Y<1 altfel X-Y

f(X,Y,REZ) :- X>(-1), Y<1, REZ is X+Y-2,!.

f(X,Y,REZ) :- REZ is X-Y.

factorial(0,1).

factorial(N,R) :- N1 is N-1, factorial(N1,R1), R is R1\*N.

**%an = 2\*(an-1)+1**

a(0,-2).

a(N,S) :- N1 is N-1, a(N1,R1), S is 2\*R1+1.

**%Fibonacci**

fib(0,0).

fib(1,1).

fib(N,S) :- N1 is N-1, N2 is N-2, fib(N1,R1), fib(N2,R2), S is R1+R2.

**%Suma de n nr**

sum(0,0).

sum(1,1).

sum(N,S) :- N1 is N-1, sum(N1,R1), S is R1+N.

f(0,-1).

f(1,2).

f(N,R) :- N1 is N-1, N2 is N-2, f(N1,R1), f(N2,R2), R is 2\*R1-3\*R2.

pow(\_,0,1).

pow(0,X,0).

pow(N,X,R) :- X1 is X-1, pow(N,X1,R1), R is R1\*N.

**%apare nr in lista?**

membru(H,[H|T]) :- !.

membru(H,[X|T]) :- membru(H,T).

**lungime([],0).**

lungime([H|T],L) :- lungime(T,L1), L is L1+1.

**%suma lista**

sum([],0).

sum([H|T], S) :- sum(T,S1), S is S1+H.

**%concatenare 2 liste - add all of list 1, then add all list 2**

%concatenare([2,1,3],[2,1,7],TR).

concatenare([],L2,L2).

concatenare([H1|T1], L2, [H1|TR]) :- concatenare(T1,L2,TR).

**%apare nr ori in lista**

count(E, [], 0).

count(E, [E|T], NR):- count(E, T, NR1), NR is NR1 + 1.

count(E, [H|T], NR):- count(E, T, NR).

**sterge(E,[],[]).**

sterge(E,[E|T],TR) :- sterge(E,T,TR).

sterge(E,[X|T],[X|TR]) :- sterge(E,T,TR).

**impartire([], [], []).**

impartire([H|T], [H|TPOZ], LNEG):- H >= 0, impartire(T, TPOZ, LNEG), !.

impartire([H|T], LPOZ, [H|TNEG]):- impartire(T, LPOZ, TNEG).

**%interclasare**

interclasare([],L2,L2).

interclasare(L1,[],L1).

interclasare([H1|T1],[H2|T2],[H1|TR]) :- H1<H2, interclasare(T1,[H2|T2],TR), !.

interclasare([H1|T1],[H2|T2],[H2|TR]) :- interclasare([H1|T1],T2,TR).

**%Divide in 2 lists based on >=k and <k**

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

divK([H|T], K, [H|TPOZ], L2) :- H>=K, divK(T, K, TPOZ, L2),!.

divK([H|T], K, L1, [H|TNEG]) :- divK(T,K,L1,TNEG).

**lungime([], 0).**

lungime([\_|T], X):- lungime(T, Y), X is Y + 1.

suma([], 0).

suma([H|T], S):- suma(T, S1), S is S1 + H.

**media\_aritmetica([], 0).**

media\_aritmetica(L, M):- suma(L, S), lungime(L, NR), M is S / NR.

**% lista**

inversare([], L, L).

inversare([H|T], Temp, I):- inversare(T, [H|Temp], I).

invers(L, I):- inversare(L, [], I).

**% inclus**

membru(E,[E|T]) :- !.

membru(E,[X|T]) :- membru(E,T).

inclus([], \_).

inclus([H|T], L2):- member(H, L2), inclus(T, L2).

**% reuniunea**

reuniune([],L2,L2).

reuniune([H|T],L2,[H|TR]) :- not(membru(H,L2)), reuniune(T,L2,TR), !.

reuniune([H|T],L2,TR) :- reuniune(T,L2,TR).

**%intersectia**

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

intersectie([H1|T1], B, [H1|TR]):- member(H1, B), intersectie(T1, B, TR), !.

intersectie([H1|T1], B, R):- intersectie(T1, B, R).

**%diferenta 2 liste**

diferenta([],L2,[]).

diferenta([H|T],L2,[H|TR]) :- not(membru(H,L2)), diferenta(T,L2,TR), !.

diferenta([H|T],L2,TR) :- dif(T,L2,TR).

**%Media aritmetica in lista**

media([],0,0,0).

media([H|T],S,L,M) :- media(T,S1,L1,M1), S is S1+H, L is L1+1, M is S/L.

max(H1,H2,H1) :- H1>H2,!.

max(\_,H2,H2).

**maxInLista([H],H).**

maxInLista([H|T],MAX) :- maxL(T,MAX2), max(H,MAX2, MAX).

maxInLista2([H],H).

maxInLista2([H|T], Max) :- maxL2(T,Max1), (H>=Max1, Max is H ; H<Max1, Max is Max1).

**%Ex1: PRODUS SCALAR 2 VECTORI**

produsVV([],[],0).

produsVV([H1|T1], [H2|T2], S) :-

produsVV(T1,T2,R1), S is H1\*H2+R1.

%produsScalar([1,2,3],[1,2,3],R).

**%Ex2: Vector\*Matrice**

produsVM(V,[],[]).

produsVM(V,[HM|TM],[R|TR]) :-

produsScalar(V,HM,R),

produsVM(V,TM,TR).

%produsVM([1,2], [[2,1],[1,3],[4,2]],R).

%[4,7,8]

**%Ex3: Matrice1\*Matrice2**

produsMM([],M2,[]).

produsMM([HM|TM],M2,[R|HR]) :-

produsVM(HM,M2,R), %fiecare linie inmultita cu matricea

produsMM(TM,M2,HR). %conditia recursiva de continuare

%produsMM([[1,2],[2,1]],[[2,1],[1,3],[4,2]],R).

%R = [[4, 7,8], [5, 5, 10]]

**%Structs**

**%firma = f(P,V,CH).**

profit([]).

profit([f(P,V,CH)|T]) :-

P is V-CH,

profit(T).

**%LISTA de profituri:**

profit2([],[]).

profit2([f(V,CH)|T], [P|ListaProfituri]) :-

P is V-CH,

profit2(T, ListaProfituri).

%profit2([f(6000,1000),f(5000,7000)], ListaProfituri).

**%listaElevi, 3 note, listaMedii.**

elevi([],[]).

elevi([e(N1,N2,N3)|T],[Media|ListaMedii]) :-

Media is (N1+N2+N3)/3,

elevi(T,ListaMedii).

%elevi([e(3,4,5),e(4,5,6)], ListaMedii)

maxLista([X], X).

maxLista([H|T], Max) :- maxLista(T, Max2), Max is max(H, Max2).

maxime([], []).

maxime([H1|H2], [T1|T2]) :- maxLista(H1, T1), maxime(H2, T2).

%maxime([[3, 5, 2], [1, 9, 8], [7, 4]], Rezultat).

concatenare([],L2,L2).

concatenare([H1|T1], L2, [H1|TR]) :- concatenare(T1,L2,TR).

concatListe([], []).

concatListe([H|T], Rezultat) :- concatListe(T, Partial), concatenare(H, Partial, Rezultat).

medii([],[]).

medii([H1|H2],[T1|T2]) :- media(H1,T1), medii(H2, T2).

%medii([[8,9,7], [10,9], [7,9,6,2,8]], LM).

**%BUBBLE SORT**

schimba([],[]).

schimba([H1,H2|T1],[H2,H1|T2]) :- H1<H2.

schimba([H|T1],[H|T2]) :- schimba(T1,T2).

%schimba([47,23,12,17,30],R). //doar 2 valori schimbate

%R = [23, 47, 12, 17, 30] .

% bsort(L,L) :- verificaSortat(L),!.

bsort(L,L).

bsort(L,LSort) :-

schimba(L,Temp), bsort(Temp, LSort).

%bsort([47,23,12,17,30],R). //sorteaza complet

%R = [12, 17, 23, 30, 47] .

**%INSERT SORT**

%47,23,12,17

%LSort =[47]

%LSort =[23,47]

%LSort=[12,23,47]

%LSort=[12,17,23,47].

insert(X,[],[X]).

insert(X,[H|T],[H|T2]) :- X>H, insert(X,T,T2), !.

insert(X,L,[X|L]).

isort([],[]).

isort([H|T],LSort) :-

isort(T,LSort),

insert(H,TSort,LSort).

**%QUICK SORT**

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

split(E,[H|T],[H|T1],T2) :- H=<E, split(E,T,T1,T2), !.

split(E,[H|T],T1,[H|T2]) :- split(E,T,T1,T2).

concatenare([],L2,L2).

concatenare([H1|T1],L2,[H1|TR]) :- concatenare(T1,L2,TR).

qsort([],[]).

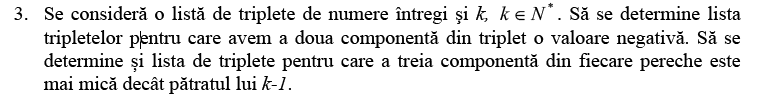
qsort([H|T],Lsort) :-

split(H,T,L1,L2),

qsort(L1,R1),

qsort(L2,R2),

concatenare(R1,[H|R2],Lsort).



**//Listele cu N2 negative excluse**

triplete([],[]).

triplete([t(N1,N2,N3)|T], [t(N1,N2,N3)|TR]) :- N2<0, triplete(T,TR), !.

triplete([t(N1,N2,N3)|T], TR) :- triplete(T,TR).

APEL:

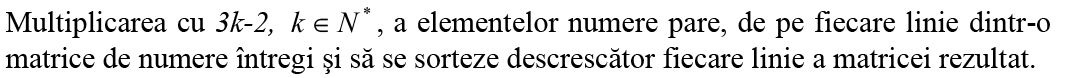
?- triplete([t(1,2,3),t(-1,3,4),t(5,-1,4),t(1,-2,3)],Liste). => Liste= [t(5, -1, 4), t(1, -2, 3)].

**//N3< (N-1) la patrat**

triplete(K,[],[]).

triplete(K,[t(N1,N2,N3)|T], [t(N1,N2,N3)|TR]) :- N3<(K-1)\*(K-1), triplete(K,T,TR), !.

triplete(K,[t(N1,N2,N3)|T], TR) :- triplete(K,T,TR).

****

**multipLL(K,[],[]).**

multipLL(K,[HL|TL],[HR|TR]) :-

multipL(K,HL,HRez),

qsort(HRez,HSortat),

inversare(HSortat,[],HR),

multipLL(K,TL,TR).

multipL(K,[],[]).

multipL(K,[H|T],[HR|TR]) :- H mod 2 =:= 0, HR is H\*(3\*K-2), multipL(K,T,TR), !.

multipL(K,[H|T], [H|TR]) :- multipL(K,T,TR).

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

split(E,[H|T],[H|T1],T2) :- H=<E, split(E,T,T1,T2), !.

split(E,[H|T],T1,[H|T2]) :- split(E,T,T1,T2).

concatenare([],L2,L2).

concatenare([H1|T1],L2,[H1|TR]) :- concatenare(T1,L2,TR).

qsort([],[]).

qsort([H|T],Lsort) :-

split(H,T,L1,L2),

qsort(L1,R1),

qsort(L2,R2),

concatenare(R1,[H|R2],Lsort).

executie(K,LL,Rezultat,RSortat) :-

multipLL(K,LL,Rezultat).

inversare([], L, L).

inversare([H|T], Temp, I):- inversare(T, [H|Temp], I).

invers(L, I):- inversare(L, [], I).

% multipLL(1,[[1,3,2],[2,4,6],[1,3,5]],Rezultat).