LISTE ÎN PROLOG

I. Definirea listelor în PROLOG

a. Lista vida: []

b. Lista nevida cu valori simboluri: [a,b,c,d,e]

c. În Prolog – definire recursiva de forma: [H|T]

[H|T] semnifică:

H- primul element din listă (Head), iar T- restul listei (Tail)

Exemplu: [1,2,3,4,5] => H=1, iar T=[2,3,4,5]

d. Declararea domeniului lista în PROLOG:

```
domains
        lista=tip_date*
```

Exemplu: lista= integer*, sau matrice=lista*

II. Prelucrări asupra listelor

1. Verificarea apartenenței unui element într-o listă de numere reale.
   (X in L, unde L=[H1, H2, H3, …, Hn])

Program PROLOG

```
domains
        lista=real*
predicates
        member(real, lista)
clauses
        member(X, [X|T]):-!.                    /* member(X, [X|_]):-!. */
        member(X, [H|T]):- member(X,T). /*member(X, [_|T]):- member(X,T). */
GOAL:   member(7, [3, 7, -2, 5, 9, 0]).        member(-7, [3, 7, -2, 5, 9, 0]).
```

2. Lungimea (dimensiunea) unei liste date.

Program PROLOG

```
domains
        lista=real*
predicates
        lungime(lista, integer)
clauses
```

```
        lungime([], 0).
        lungime([H|T], L):- lungime(T, L1), L=L1+1.
```

```
GOAL: lungime([3, 7, -2, 5, 9, 0], Lungime).
```

3. **Suma elementelor dintr-o listă de numere întregi.**

Program PROLOG

```
domains
        lista=integer*
predicates
        suma(lista, real)
clauses
        suma([], 0).
        suma([H|T], S):- suma(T, S1), S=S1+H.
```

```
GOAL: suma([3, 7, -2, 5, 9, 0], Suma).
```

4. **Concatenarea (alipirea) a două liste de numere reale (LR=L1L2).**
   L1=[H1,H2,…,Hn], L2=[Z1,Z2,…,Zm], LR=[H1,H2,…,Hn, Z1,Z2,…,Zm] cu precizarea LR=[HR|TR], unde HR=H1, iar TR=[T1,L2], T1=restul listei L1.

Program PROLOG

```
domains
        lista=integer*
predicates
        concatenare(lista, lista, lista)
clauses
        concatenare([], L2, L2).
        concatenare([H1|T1], L2, [H1|TR]):- concatenare(T1, L2, TR).
```

```
GOAL: concatenare([3,7,-2,5], [3,4,5,6], Rez).
```

5. **Numărul de apariții al unui element într-o listă de numere întregi.**

Program PROLOG

```
domains
        lista=integer*
predicates
        count(integer, lista, integer)
clauses
        count(X, [], 0).
```

```
count(X, [X|T], R):- count(X, T, R1), R=R1+1, !.
count(X, [H|T], R):- count(X, T, R).
```

GOAL: count(7, [3,7,-2,7,7,0,7,7], Rez).

6. **Ștergerea unui element dintr-o listă de numere reale.**
   a. **ștergerea primei apariții în lista dată**

```
sterge1(X, [X|T], T):-!.
sterge1(X, [Y|T], [Y|T1]):- sterge1(X, T, T1).
```

   b. **ștergerea tuturor aparițiilor în lista dată**

```
sterge2(X, [], []).
sterge2(X, [X|T], R):- sterge2(X,T,R), !.
sterge2(X, [Y|T], [Y|TR]):- sterge2(X, T, TR).
```

GOAL: sterge1(7, [3,7,-2,7,7,0,7,7], Rez) sterge2(7, [3,7,-2,7,7,0,7,7], Rez)

7. **Împărțirea unei liste de numere reale în două liste, o listă cu valori pozitive, iar cealaltă cu valori negative.**

Program PROLOG

```
domains
        lista=real*
predicates
        impartire(lista, lista, lista)
clauses
  impartire([], [], []).
  impartire([H|T], [H|TPoz], LNeg):- H>0, impartire(T, TPoz, LNeg), !.
  impartire([H|T], LPoz, [H|TNeg]):- impartire(T, LPoz, TNeg).
```

GOAL: impartire([3,7,-2,7,-7,0,7,-8], LPoz, LNeg)

III. Aplicații folosind listele

1. **Împărțirea unei liste de numere reale în două liste în funcție de un parametru real constant k.**

Program PROLOG

```
domains
        lista=real*
predicates
```

```
        impartirek(real, lista, lista, lista)
    clauses
      impartirek(K, [], [], []).
      impartirek(K, [H|T], [H|Tmari], Lmici):- H>K, impartirek(K, T,
Tmari, Lmici), !.
      impartirek(K, [H|T], Lmari, [H|Tmici]):- impartirek(K, T, Lmari,
Tmici).
```

GOAL: impartirek(7, [3,17,-2,7,-7,0,9,-8], LKmari, LKmici)

2. **Media aritmetică a valorilor dintr-o listă de numere reale.**

```
/* media_aritmetica(lista, real)        */
 media_aritmetica(L, Ma):- suma(L, S), lungime(L, N), Ma= S/N.
suma([],0).
suma([H|T],S):- suma(T, St), S is St+H.
lungime([],0).
lungime([H|T],N):- lungime(T, N1), N is N1+1.
Goal: media_aritmetica([4,7,8,-5,9], M).
```

3. **Să se determine reversa (inversa) unei liste de numere întregi.**
   (Observație: Reversa([3 5 7 9])= [9, 7, 5, 3])

Program PROLOG

```
    domains
        lista=integer*
    predicates
        reversa(lista, lista)
        concatenare(lista,lista,lista)
    clauses
        concatenare([], L2, L2).
        concatenare([H1|T1], L2, [H1|TR]):- concatenare(T1, L2, TR).
        reversa([], []).
        reversa([H|T], R):- reversa(T, R1), concatenare(R1,[H],R).
```

GOAL: reversa([3, 7, -2, 5, 9, 0], Rev).

4. **Să se calculeze suma numerelor pare/impare dintr-o listă de numere întregi.**

Program PROLOG

```
domains
        lista=integer*
predicates
        sumapar(lista, integer)
        sumaimpar(lista, integer)
clauses
        sumapar([],0).
        sumapar([H|T], Spar):- H mod 2 =0, sumapar(T,R), Spar=R+H.
        sumapar([H|T], Spar):- H mod 2 =1, sumapar(T, Spar).
        sumaimpar([],0).
        sumaimpar([H|T], Simpar):- H mod 2 =1, sumaimpar(T,R),
Simpar=R+H.
        sumaimpar([H|T], Simpar):-  H mod 2 =0, sumaimpar(T,Simpar).
```

GOAL: sumapar([3, 7, -2, 5, 9, 0], Spar) | sumaimpar([3, 7, -2, 5, 9], Simpar)

Suma numerelor pare și impare dintr-o listă de numere întregi.

```
sumaparimpar([],0,0).
sumaparimpar([H|T], Spar,Simpar):- H mod 2=0,
sumaparimpar(T,R1,Simpar), Spar=R1+H.
sumaparimpar([H|T], Spar,Simpar):- H mod 2 =1,
sumaparimpar(T,Spar,R2), Simpar=R2+H.
```

GOAL: sumaparimpar([3, 7, -2, 6, 19, 0], Spar, Simpar)

5. **Concatenarea a 3 liste de numere reale.**

```
/* concatenare3(lista, lista, lista, lista)    */
/* concatenare3(L1,L2,L3,R):- concatenare(L1, L2, R1), concatenare(R1,
L3, R).        */
concatenare([],L,L).
concatenare([H|T],L2,[H|Tr]):-concatenare(T,L2,Tr).
```

GOAL: concatenare3([3, 7], [2,5], [0,1], Rez).

6. **Să se determine reuniunea a două mulțimi reprezentate prin liste.**

```
/* reuniune(lista, lista, lista)        */
/* reuniune(L1,L2,R):- ?.        */
```

Program PROLOG

```
domains
        lista=integer*
predicates
        member(integer, lista)
        reuniune(lista, lista, lista)
clauses
        member(X, [X|T]):-!.
        member(X, [_|T]):- member(X, T).
        reuniune([], L2, L2).
        reuniune([H|T], L2, [H|TR]):- not(member(H, L2)), reuniune(T,
L2, TR), !.
        reuniune([H|T], L2, R):- reuniune(T, L2, R).
```

GOAL: reuniune([3, 7, -2, 5], [5, 7, 2], Reuniune)

7. **Să se determine intersecția a două mulțimi reprezentate prin liste.**

```
/* intersectie(lista, lista, lista)      */
/* intersectie(L1,L2,R):- ?.        */
```

8. **Să se determine maximul elementelor dintr-o listă de numere reale.**

Program PROLOG

```
domains
        lista=integer*
predicates
        maxim(integer, integer, integer)
        maximL(lista, integer)
clauses
        maxim(A, B, A):-A>B, !.
        maxim (A, B, B).

        maximL([X], X).
        maximL([H|T], Max):-maximL(T, MaxT), maxim(H, MaxT,
Max).
```

GOAL: maximL([3, 7, -2, 5, 15, 7, 2], MaxLista)

9. **Să se determine maximul elementelor dintr-o matrice de numere intregi.**

Program PROLOG

```
domains
```

```
        lista= integer*
        matrice= lista*
    predicates
        maxim(integer, integer, integer)
        maximL(lista, integer)
        maximM(matrice, integer)
    clauses
        maxim(A, B, A):-A>B, !.
        maxim (A, B, B).

        maximL([X], X).
        maximL([H|T], Max):-maximL(T, MaxT), maxim(H, MaxT,
Max).
        maximM([X], M):- maximL(X, M).
        maximM([H|T], MaxM):-maximM(T, MaxT1),  maximL(H,
MaxT2), maxim(MaxT1, MaxT2, MaxM).
```

GOAL: maximM([[3, 7, -2], [5, 15, 7], [2, 3, -2]], MaxMatrix)

10. Să se calculeze produsul scalar a doi vectori de numere întregi.

```
/* produs_scalar(lista, lista, real) */
/* produs_scalar([],[],0).        */
/* produs_scalar([H1|T1],[H2|T2], R):- produs_scalar(T1,T2,R1),
R=R1+H1*H2.     */
```

GOAL: produs_scalar([3, 7, -2], [5, 1, 2], Produs_scalar)

11. Înmulțirea unui vector cu o matrice.

```
/* inmultireVM(lista, matrice, lista)       */
/* inmultireVM(V,[],[]).        */
/* inmultireVM(V,[Hm|Tm], [Hr|Tr]):- produs_scalar(V,Hm,Hr),
inmultireVM(V, Tm, Tr).     */
```

GOAL: inmultireVM([1, 0, -1], [[5, 1, 2], [1, 2, 0], [3, 1, 1]], Result_vector)

12. Să se determine produsul a două matrici.

```
domains
        lista= integer*
        matrice= lista*
predicates
```

```
produs_matrice(matrice,matrice,matrice)
clauses
/* inmultireVM(V,[],[]).        */
/* inmultireVM(V,[Hm|Tm], [Hr|Tr]):- produs_scalar(V,Hm,Hr),
inmultireVM(V, Tm, Tr).     */
produs_matrice([],M,[]).
produs_matrice([H1|T1],M2,[X|R]):-inmultireVM(H1,M2,X),
produs_matrice(T1,M2,R).
```

GOAL: produs_matrice([[1, 0],[2,-1]], [[1, 2], [2, 0]], Res_matrix)

13. Sa se determine lista sumelor elementelor de pe fiecare linie dintr-o matrice si sa se sorteze lista rezultat.

```
sumaL([],0).
sumaL([H|T],S):- sumaL(T,S1), S is S1+H.
sumaM([],[]).
sumaM([H|T],[SH|ST]):- sumaL(H,SH), sumaM(T,ST).
sort_line_matrix(M,LS):- sumaM(M,R), qsort(R,LS).
sort_line_matrix2(M,LS):-sumaM(M,R), bsort(R,LS).
sort_line_matrix3(M,LS):-sumaM(M,R), insertsort(R,LS).
insertsort([],[]).
insertsort([H|T],LS):- insertsort(T,LS1), insert(H,LS1,LS).
insert(H,[Y|T],[Y|T1]):- H>Y, insert(H,T,T1), !.
insert(H,L,[H|L]).
bsort(L,LS):- schimba(L,L1), bsort(L1,LS),!.
bsort(L,L).
schimba([X,Y|T],[Y,X|T]):- X>Y.
schimba([Y|T],[Y|T1]):- schimba(T,T1).
qsort([],[]).
qsort([H|T],LS):- split(H,T,L1,L2), qsort(L1,S1), qsort(L2,S2),
concatenare(S1,[H|S2],LS).
split(K,[],[],[]).
split(K,[H|T],[H|T1],L2):-H<K, split(K,T,T1,L2), !.
split(K,[H|T],L1,[H|T2]):- split(K,T,L1,T2), !.
concatenare([],L,L).
concatenare([H|T], L2, [H|TR]):- concatenare(T,L2,TR).
GOAL:- sort_line_matrix2([[2,3],[-1,8],[8,-12]],LS).
```

14. Să se determine concatenarea a N liste de numere întregi si sortarea listei rezultat !!!)

```
concatenare([],L,L).
concatenare([H|T], L2, [H|TR]):- concatenare(T,L2,TR).
concatenareN([],[]).
concatenareN([Hm|Tm],R):-concatenareN(Tm,RT), concatenare(Hm,
RT, R).
sort_matrix(M,LS):- concatenareN(M,R), qsort(R,LS).
sort_matrix2(M,LS):-concatenareN(M,R), bsort(R,LS).
sort_matrix3(M,LS):-concatenareN(M,R), insertsort(R,LS).
insertsort([],[]).
insertsort([H|T],LS):- insertsort(T,LS1), insert(H,LS1,LS).
insert(H,[Y|T],[Y|T1]):- H>Y, insert(H,T,T1), !.
insert(H,L,[H|L]).
bsort(L,LS):- schimba(L,L1), bsort(L1,LS),!.
bsort(L,L).
schimba([X,Y|T],[Y,X|T]):- X>Y.
schimba([Y|T],[Y|T1]):- schimba(T,T1).
qsort([],[]).
qsort([H|T],LS):- split(H,T,L1,L2), qsort(L1,S1), qsort(L2,S2),
concatenare(S1,[H|S2],LS).
split(K,[],[],[]).
split(K,[H|T],[H|T1],L2):-H<K, split(K,T,T1,L2), !.
split(K,[H|T],L1,[H|T2]):- split(K,T,L1,T2), !.
concatenare([],L,L).
concatenare([H|T], L2, [H|TR]):- concatenare(T,L2,TR).
GOAL:- sort_matrix([[2,3],[-1,8],[8,-12]],LS).
GOAL:- concatenareN([[2,3],[-1,8],[7,6,2]], R).
```

15. Sa se determine lista maximelor pe fiecare linie dintr-o matrice si sa se sorteze descrescator lista obtinuta.

```
maxim(A,B,A):-A>B,!.
maxim(A,B,B).
maxim_lista([W],W).
maxim_lista([H|T],Max):- maxim_lista(T,MaxT), maxim(H,MaxT,Max).
calcul_lista_maxime([],[]).
```

```
calcul_lista_maxime([H|T],[Hmax|Tmax]):-
maxim_lista(H,Hmax),calcul_lista_maxime(T,Tmax).
sort_maxim_matrix(M,LMS):- calcul_lista_maxime(M,LMax),
qsort(LMax,LMS).
sort_maxim_matrix(M,LMS):- calcul_lista_maxime(M,LMax),
bsort(LMax,LMS).
sort_maxim_matrix(M,LMS):- calcul_lista_maxime(M,LMax),
insertsort(LMax,LMS).
insertsort([],[]).
insertsort([H|T],LS):- insertsort(T,LS1), insert(H,LS1,LS).
insert(H,[Y|T],[Y|T1]):- H>Y, insert(H,T,T1), !.
insert(H,L,[H|L]).
bsort(L,LS):- schimba(L,L1), bsort(L1,LS),!.
bsort(L,L).
schimba([X,Y|T],[Y,X|T]):- X>Y.
schimba([Y|T],[Y|T1]):- schimba(T,T1).
qsort([],[]).
qsort([H|T],LS):- split(H,T,L1,L2), qsort(L1,S1),
qsort(L2,S2),concatenare(S1,[H|S2],LS).
split(K,[],[],[].
split(K,[H|T],[H|T1],L2):-H<K, split(K,T,T1,L2), !.
split(K,[H|T],L1,[H|T2]):- split(K,T,L1,T2), !.
concatenare([],L,L).
concatenare([H|T], L2, [H|TR]):- concatenare(T,L2,TR).
GOAL:- sort_maxim_matrix([[2,3],[-1,8],[8,-12]],LS).
```

16. Sa se calculeze lista valorilor cmmdc pentru liniile dintr-o matrice de numere intregi.

```
cmmdc()
cmmdcL()
cmmdcM()
```

GOAL: cmmdcM([[10, 5], [20,-10], [10, 20, -20, 10]], Lcmmdc)

17. Sa se calculeze valorile profit pentru o lista de firme pentru care se retin valorile – venituri si cheltuieli (P=V-Ch).

```
calcul_profit([]).
```

```
calcul_profit([f(V,Ch,P)|TFirme]):-P is V-Ch, calcul_profit(TFirme).
```

GOAL: calcul_profit([f(4000,2000,P1), f(2000,1000,P2)]).

18. Sa se calculeza valorile profit pentru o lista de firme, sa se determine lista valorilor profit si sa se sorteze lista valorilor profit.

```
calcul_profitL([],[]).
calcul_profitL([h(V,Ch)|TFirme],[P|TP]):- P is V-Ch,
calcul_profitL(Tfirme, TP).
qsort([],[]).
qsort([H|T],LS):- split(H,T,L1,L2), qsort(L1,S1), qsort(L2,S2),
concatenare(S1, [H|S2], LS).
split(K,[],[],[]).
split(K,[H|T],[H|T1],L2):- H<K, split(K,T, T1, L2), !.
split(K,[H|T],L1,[H|T2]):- split(K,T, L1, T2).
concatenare([],L2,L2).
concatenare([H|T],L2,[H|TR]):-concatenare(T,L2,TR).
profit_list_sort(Lf, LSP):- calcul_profitL(Lf,LP), qsort(LP, LSP).
```

GOAL: profit_list_sort([h(8000,2000), h(5000,1000)], LSP).

19. Se considera lista de note n(n1,n2,n3). Sa se calculeze mediile corespunzatoare intr-o lista separata (lista de medii).

```
list_medii([],[]).
list_medii([n(N1,N2,N3)|Tn],[HM|TM]):- HM is (N1+N2+N3)/3,
list_medii(Tn,TM).
```

GOAL: list_medii([n(8,9,10), n(8,8,10)], LM).

20. Sa se calculeze mediile corespunzatoare dar prin excluderea notelor minime.

```
list_medii2([],[]).
list_medii2([n(N1,N2,N3)|Tn],[HM|TM]):- minim(N1,N2,N3,Min),
 HM is (N1+N2+N3-Min)/2, list_medii2(Tn,TM).
min(A,B,B):-A>B,!.
min(A,B,A).
minim(A,B,C,Min):- min(A,B,R1), min(R1,C,Min).
```

GOAL: list_medii2([n(8,10,10), n(9,10,10)], LM2).

21. Sa se calculeze mediile corespunzatoare notelor obtinute de elevii unei clase. (Ind: se poate utiliza o matrice, fiecare linie din matrice va retine lista notelor pentru un anumit elev).

```
calcul_medii_elevi([[8,9,7], [10,9], [7,9,6,2,8]], LM)
```

22. Sa se determine lista mediilor de promovare, respectiv lista mediilor elevilor nepromovati pentru cerinta 21.