

P1: Liste in Prolog (1)

1.

- a. Sa se scrie un predicat care sterge toate aparitiile unui anumit atom dintr-o lista.
- b. Definiti un predicat care, dintr-o lista de atomi, produce o lista de perechi (atom n), unde atom apare in lista initiala de n ori. De ex: `numar([A B A B A C A], X)` va produce `X = [[A 4] [B 2] [C 1]]`.

```
domains
  lista=integer*
  perechi=lista*
predicates
  sterge(integer,lista,lista)
  nr_apar(integer,lista,integer)
  get_perechi(lista,perechi)
clauses
  sterge(_,[],[]).
  sterge(E,[H|T],[H|T1]):-H<>E,!,sterge(E,T,T1).
  sterge(E,[_|T],T1):-sterge(E,T,T1).
  nr_apar(_,[],0).
  nr_apar(E,[E|H],N):-nr_apar(E,H,N1),N=N1+1,!.
  nr_apar(E,[_|H],N):-nr_apar(E,H,N).
  get_perechi([],[]).
  get_perechi([E],[[E,1]]).
  get_perechi([H|T],[[H,N]|P]):-nr_apar(H,[H|T],N),
                                sterge(H,[H|T],L),
                                get_perechi(L,P).
```

2.

- a. Sa se scrie un predicat care intoarce reuniunea a doua multimi.

```
domains
  lista=integer*
predicates
  apartine(integer,lista)
  reuniune(lista,lista,lista)
clauses
  apartine(E,[E|T]).
  apartine(E,[H|T]):-apartine(E,T).
  reuniune([],T1,T1).
  reuniune([H2|T2],L2,[H2|T1]):-not(apartine(H2,L2)),!,reuniune(T2,L2,T1).
  reuniune([_|T2],L2,T1):-reuniune(T2,L2,T1).
```

- b.** Sa se scrie un predicat care, primind o lista, intoarce multimea tuturor perechilor din lista. De ex, cu [a b c d] va produce [[a b] [a c] [a d] [b c] [b d] [c d]].

```
domains
    lista=integer*
    listad=lista*
predicates
    pereche(integer,lista,listad)
    member(integer,lista)
    gasesc(lista,listad)
    princ(lista,listad)
    concat1(listad,listad,listad)
clauses
    %gasesc([],L):-!.
    % gasesc([A1|L1],[[A1|L1]|_]):- !.
    % gasesc([A1|L1],[_|L]):-gasesc([A1|L1],L).
    gasesc([A1,A2],[[A1,A2]|_]):-!.
    gasesc([A1,A2],[[A2,A1]|_]):-!.
    gasesc([A1,A2],[_|L]):-gasesc([A1,A2],L).
    member(E,[E|_]):-!.
    member(E,[_|T]):-member(E,T).
    CONCAT1([],[],[]):-!.
    concat1(L1,[],L1):-!.
    concat1([],L2,L2):-!.
    concat1([A1|L1],L2,[A1|L3]):-concat1(L1,L2,L3),NOT (GASESC(A1,L3)),!.
    concat1(L1,[A2|L2],[A2|L3]):-concat1(L1,L2,L3),!.
    pereche(_,[],[]):-!.
    pereche(A,[A|L],L2):-pereche(A,L,L2),!.
    pereche(A,[A1|L],[[A,A1]|L2]):-PERECHE(A,L,L2),
        not(gasesc([A,A1],L2)).

    princ([],[]):-!.
    princ([A|L],L2):-pereche(A,L,L4),
        princ(L,L3),
        concat1(L4,L3,L2).
```

3.

- a.** Sa se intercaleze un element pe pozitia a n-a a unei liste.

```
domains
    list=integer*
predicates
    insereaza(list,integer,integer,list)/ * integer1-pozitia */
    lucru(list, integer, integer, list)
clauses
```

```

insereaza(L, Poz, Elem, Lrez):-
    lucru(L, Poz, Elem, Lrez).

lucru([], 0, Elem, [Elem]):-!.
lucru(L, Poz, Elem, Lrez):-
    Poz = 0,
    Lrez = [Elem|L].
lucru([H|T], Poz, Elem, Lrez):-
    N = Poz-1,
    lucru(T, N, Elem, Lnou),
    Lrez = [H|Lnou],!.

```

b. Definiti un predicat care intoarce cel mai mare divizor comun al numerelor dintr-o lista.

```

domains
    list=integer*
predicates
    cmmdc(integer,integer,integer)/cmmdc(a,b,rez)*/
    cmmdcList(list,integer)
clauses
    cmmdc(A, B, Rez):-
        A = B,
        Rez = B,!.
    cmmdc(A, B, Rez):-
        B>A,
        C = B-A,
        cmmdc(C, A, Rez).
    cmmdc(A, B, Rez):-
        A>B,
        C = A-B,
        cmmdc(C, B, Rez).
    cmmdcList([H],H):-!.
    cmmdcList([H1|[H2|T]],Rezult):-
        cmmdc(H1,H2,Rez),
        cmmdcList([Rez|T],Rezult).

```

4.

- a.** Sa se scrie un predicat care substituie intr-o lista un element prin altul.
- b.** Sa se construiasca sublista (lm, ..., ln) a listei (l1, ..., lk).

```

domains
    list=integer*
predicates
    length(list,integer)
    subst(list,integer,integer,list)

```

```

        sublist(list, integer, integer, list)
clauses
    length([], 0).
    length([_|T], R) :- length(T, R1), R = R1 + 1.

    subst([], _, _, []) :- !.
    subst(L, X, _, []) :- !, length(L, Y), X > Y, !.
    subst([H|T], X, E, [H|C]) :- X < > 1, !, X1 = X - 1, subst(T, X1, E, C).
    subst([_|T], X, E, [E|T]) :- X = 1, !.

    sublist([], _, _, []).
    sublist(L, X, _, []) :- !, length(L, Y), X > Y, !.
    sublist(L, _, X, []) :- !, length(L, Y), X > Y, !.
    sublist(_, X, Y, []) :- !, X > Y, !.
    sublist([_|T], X, Y, L) :- X > 1, !, X1 = X - 1, Y1 = Y - 1, sublist(T, X1, Y1, L).
    sublist([H|T], X, Y, [H|C]) :- Y > 0, !, Y1 = Y - 1, sublist(T, X, Y1, C).
    sublist(L, _, _, L).

```

5.

- a. Sa se scrie un predicat care se va satisface daca o lista are numar par de elemente si va esua in caz contrar, fara sa se numere elementele listei.
- b. Sa se elimine prima aparitie a elementului minim dintr-o lista de numere intregi.

```

domains
    el = integer
    list = el*

```

predicates

```

%pct a
lung(list)

```

```

%pct b
min(el, el, el)    %(i, i, o)
minim(list, el)    %(i, o)
elimin(list, list) %(i, o)

```

clauses

```

%pct a
lung([]).
lung([H|[H1|T]]) :- lung(T).

```

```

%pct b
min(A, B, A) :- A < B.
min(A, B, B) :- B < A.

```

```

minim([H|T],H):-T=[],!.
minim([H|T],M):-minim(T,M1),
                min(H,M1,M).

elimin([],[]).
elimin([H|T],[]):-T=[],!.
elimin([H|T],T):- minim(T,M),H<=M,!.
elimin([H|T],[H|L]):-elimin(T,L).

```

6.

a. Sa se scrie un predicat care intoarce intersectia a doua multimi.

```

% Intersectia a doua multimi

domains
    multime=integer*

predicates
    intersectie(multime,multime,multime)
    apartine(integer,multime)

clauses
    apartine(E,[E|_]).
    apartine(E,[_|T]):- apartine(E,T).

    intersectie([],_,[]).
    intersectie([X|M1],M,[X|M2]):- apartine(X,M),
                                    intersectie(M1,M,M2).
    intersectie([X|M1],M,M2):- not(apartine(X,M)),
                                intersectie(M1,M,M2).

```

b. Sa se construiasca lista (m, ..., n), adica multimea numerelor intregi din intervalul [m, n].

```

domains
    lista=integer*

predicates
    interval(integer,integer,lista)

clauses
    interval(M,N,L) :- M>N, L=[].
    interval(M,N,L) :- M=N, L=[M].
    interval(M,N,[H|T]) :- M<N, H=M, M2=M+1,

```

interval(M2,N,T).

7.

- a. Sa se scrie un predicat care transforma o lista intr-o multime, in ordinea primei aparitii. Exemplu: [1,2,3,1,2] e transformat in [1,2,3].

```
domains
  list=integer*
predicates
  este(list,integer)
  elimin(list,integer,list)
  multime(list,list)
clauses
  este([X|_],X).
  este([_|T],X) :- este(T,X).

  elimin([],_,[]).
  elimin([M|T],M,X) :- elimin(T,M,X).
  elimin([H|T],M,[H|X]) :- elimin(T,M,X).

  multime([],[]).
  multime([H|T],[H|X]) :- elimin(T,H,Y), multime(Y,X).
goal
  multime([1,2,4,2,3,1,4,3,2],X),
  write(X)
```

- b. Sa se scrie o functie care descompune o lista de numere intr-o lista de forma [lista-de-numere-pare lista-de-numere-impare] (deci lista cu doua elemente care sunt liste de intregi), si va intoarce si numarul elementelor pare si impare.

```
domains
  list=integer*
  list2=list*
predicates
  separ(list,list2)
clauses
  separ([],[],[]).
  separ([H|T],[[H|X],Y]) :- H mod 2 = 0, !, separ(T,[X,Y]).
  separ([H|T],[X,[H|Y]]) :- separ(T,[X,Y]).
```

```
%a.Transforma lista in multime in ordinea primei
%aparitii
%b.Descompune lista in [[nr.pare][nr.impare]] si
%returneaza nr. elem pare si nr.elem impare
domains
  lista=integer*
```

```

llista=lista*
predicates
member(integer,lista)
elimina(integer,lista,lista) %elimina un elem. din lista
multime(lista,lista)
pare(lista,lista,integer) %lista elem. pare
impare(lista,lista,integer) %lista elem. impare
descomp(lista,llista,integer,integer)
clauses
member(A,[A|_]):-!.
member(A,[_|T]):-member(A,T).
elimina(A,[],[]).
elimina(A,[A|T],T1):-elimina(A,T,T1),!.
elimina(A,[H|T],[H|T1]):-elimina(A,T,T1).
multime([],[]).
multime([H|T],[H|T1]):-member(H,T),!,elimina(H,T,T2),
    multime(T2,T1).
multime([H|T],[H|T1]):-multime(T,T1).
pare([],[],0).
pare([H|T],[H|T1],N):-H mod 2=0,!,pare(T,T1,N1),
    N=N1+1.
pare([_|T],T1,N):-pare(T,T1,N).
impare([],[],0).
impare([H|T],[H|T1],N):-H mod 2<>0,!,impare(T,T1,N1),
    N=N1+1.
impare([_|T],T1,N):-impare(T,T1,N).
descomp([],[],0,0):-!.
descomp([A],[A],1,0):-A mod 2=0,!.
descomp([A],[A],0,1):-A mod 2<>0,!.
descomp(L,[L1,L2],N1,N2):-pare(L,L1,N1),
    impare(L,L2,N2).

```

8.

- a. Sa se scrie un predicat care intoarce diferenta a doua multimi.
- b. Sa se scrie un predicat care adauga intr-o lista dupa fiecare element par valoarea 1.

```

domains
    multime,lista=integer*
predicates
diferenta(multime,multime,multime)
member(integer,multime)
adauga(lista,lista)
clauses
member(E,[E|_]):-!.
member(E,[_|T]):-member(E,T).
diferenta(L,[],L):-!.
diferenta([],_,[]).
diferenta([H|T],L,[H|T1]):-not(member(H,L)),!,
    diferenta(T,L,T1).
diferenta([_|T],L,L1):-diferenta(T,L,L1).

```

```

adauga([],[]).
adauga([H|T],[H|[1|T1]]):-H mod 2=0,!,
    adauga(T,T1).
adauga([H|T],[H|T1]):-adauga(T,T1).

```

9.

- a.** Sa se scrie un predicat care transforma o lista intr-o multime, in ordinea ultimei aparitii. Exemplu: [1,2,3,1,2] e transformat in [3,1,2].
- b.** Sa se calculeze cel mai mare divizor comun al elementelor unei liste.

domains

```

    el = integer
    list = el*

```

predicates

```

    % adauga elementul la sfarsitul listei
    adaug_sf(integer, list, list)
    % inverseaza o lista
    invers(list, list)
    % verifica daca un element apartine sau nu listei
    member(el, list)
    % transforma o multime intr-o lista
    multime(list, list)

    % calculeaza cmmdc al 2 numere
    nondeterm cmmdc(el, el, el)
    nondeterm cmmdcl(list, el)

```

clauses

```

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

    invers([], []).
    invers([H|T], L) :- invers(T, L1), adaug_sf(H, L1, L).

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

    multime([], []).
    multime([H|T], [H|L]) :- not(member(H, T)), !, multime(T, L).
    multime([_|T], L) :- multime(T, L).

    cmmdc(1, _, 1) :- !.
    cmmdc(_, 1, 1) :- !.
    cmmdc(0, B, B) :- !.
    cmmdc(B, 0, B) :- !.
    cmmdc(X, X, X) :- !.
    cmmdc(A, B, L) :- A < B, !, L1 = B mod A, cmmdc(A, L1, L).
    cmmdc(A, B, L) :- L1 = A mod B, cmmdc(B, L1, L).

    cmmdcl([], 0).
    cmmdcl([H|T], C) :- cmmdcl(T, D), cmmdc(D, H, C).

    % invers([1, 2, 3], L).
    % multime([1, 2, 3, 1, 2, 4, 5, 4, 6, 9], X).
    % cmmdcl([64, 24], X).

```


10.

- a. Sa se scrie un predicat care determina cel mai mic multiplu comun al elementelor unei liste formate din numere intregi.
- b. Sa se scrie un predicat care adauga dupa 1-ul, al 2-lea, al 4-lea, al 8-lea ...element al unei liste o valoare v data.

domains

el = integer
list = el*

predicates

```
%pct a
cmmdc(el,el,el)  %(i,i,o)
cmmmc(el,el,el)  %(i,i,o)
cmmmcL(list,el)  %(i,o)

%pct b
adaug(list,integer,list)  %(i,i,o)
adaug_aux(list,integer,integer,integer,list)  %(i,i,i,o)
```

clauses

```
%pct a
cmmdc(A,A,A).
cmmdc(A,B,D):-A>B,!,A1=A-B,cmmdc(A1,B,D).
cmmdc(A,B,D):-B>A,!,B1=B-A,cmmdc(A,B1,D).

cmmmc(A,B,M):-P=A*B,cmmdc(A,B,D),M=P/D.

cmmmcL([],1).
cmmmcL([H|T],M):-cmmmcL(T,M1),cmmmc(H,M1,M).

%pct b
adaug(L1,V,L2):-Poz = 1,
                Con = 1,
                adaug_aux(L1,V,Poz,Con,L2).

adaug_aux([],_,_,_,[]).
adaug_aux([H|T],V,Poz,Con,[H|[V|L]]):-Poz=Con,!,
                Poz2=Poz*2,
                Con2=Con+1,
                adaug_aux(T,V,Poz2,Con2,L).
adaug_aux([H|T],V,Poz,Con,[H|L]):-not(Poz=Con),!,
                Con2=Con+1,
                adaug_aux(T,V,Poz,Con2,L).
```

11.

a. Sa se scrie un predicat care testeaza daca o lista este multime.

```
domains
  list=integer*
predicates
  multime(list)
  member(integer,list)
clauses
  member(E,[E|_]):-!.
  member(E,[_|T]):-member(E,T).
  multime([]).
  multime([_]):-!.
  multime([H|T]):-not (member(H,T)),multime(T).
```

b. Sa se scrie un predicat care elimina primele 3 aparitii ale unui element intr-o lista. Daca elementul apare mai putin de 3 ori, se va elimina de cate ori apare.

```
domains
  el = integer
  list = el*

predicates

  %pct b
  elimin(i,list,list)  %(i,i,o)
  eliminare(el,integer,list,list)  %(i,i,i,o)

clauses

  %pct b
  elimin(E,L1,L2):-eliminare(E,3,L1,L2).

  eliminare(_,_,[],[]).
  eliminare(E,Ap,[E|T],L):-Ap=1,!,
    L=T.
  eliminare(E,Ap,[E|T],L):-Ap>1,Ap2=Ap-1,!,
    eliminare(E,Ap2,T,L).
  eliminare(E,Ap,[H|T],[H|L]):-eliminare(E,Ap,T,L).
```

12.

- a. Sa se scrie un predicat care testeaza egalitatea a doua multimi, fara sa se faca apel la diferenta a doua multimi.
- b. Definiti un predicat care selecteaza al n-lea element al unei liste.

```
domains
  el=integer
  list=el*
predicates
  aln(list,el,el)
  mult(list,list)
  member(el,list)
  egal(list,list)
clauses
  aln([H|T],1,P):- P=H.
  aln([H|T],N,P):- N1=N-1,aln(T,N1,P).
  member(E,[E|_]):-!.
  member(E,[_|T]):-member(E,T).
  mult([],L):-!.
  mult([H|T],L):-member(H,L),mult(T,L).
  egal(L1,L2):- mult(L1,L2),mult(L2,L1).
```

13.

- a. Sa se scrie un predicat care substituie intr-o lista un element printr-o alta lista.
- b. Sa se elimine elementul de pe pozitia a n-a a unei liste liniare.

```
domains
  el = integer
  list = el*

predicates

  %pct a
  concatenare(list,list,list)  %(i,i,o)
  subst(el,list,list,list)  %(i,i,i,o)

  %pct b
  elimin(list,integer,list)  %(i,i,o)

clauses

  %pct a
  concatenare([],L,L).
  concatenare([H|L1],L2,[H|L3]):-concatenare(L1,L2,L3).

  subst(_,_,[],[]).

  subst(E,L,[E|T],R):-subst(E,L,T,R1),!,
    concatenare(L,R1,R).
```

```
subst(E,L,[H|T],[H|R]):-subst(E,L,T,R).
```

```
%pct b
elimin([],_,[]).
elimin([H|T],1,T):-!.
elimin([H|T],N,[H|L]):-N1=N-1,
                        elimin(T,N1,L).
```

14.

- a. Sa se scrie un predicat care elimina dintr-o lista toate elementele care se repeta (ex: $l=[1,2,1,4,1,3,4] \Rightarrow l=[2,3]$)
- b. Sa se elimine toate aparitiile elementului maxim dintr-o lista de numere intregi.

```
domains
  el = integer
  list = el*
```

```
predicates
  %pct a
  member(el,list)
  eliminNr(list,list)  %(i,o)

  %pct b
  maxim(list,el)  %(i,o)
  eliminMax(list,list)  %(i,o)
  elimin(el,list,list)  %(i,i,o)
```

clauses

```
%pct a
member(E,[E|_]).
member(E,[_|T]):-member(E,T).

eliminNr([],[]).
eliminNr([H|T],L):-member(H,T),!,
                  elimin(H,T,L1),
                  eliminNr(L1,L).
eliminNr([H|T],[H|L]):-eliminNr(T,L).
```

```
%pct b
maxim([],-3200).
maxim([H|T],M):-maxim(T,M),H<M,!.
maxim([H|_],H).

elimin(E,[],[]).
```

```

elimin(E,[E|T],L):-!,elimin(E,T,L).
elimin(E,[H|T],[H|L]):-elimin(E,T,L).

```

```

eliminMax([],[]).
eliminMax(L1,L2):-maxim(L1,M),
                  elimin(M,L1,L2).

```

15.

- a. Sa se scrie un predicat care sa testeze daca o lista formata din numere intregi are aspect de "vale"(o multime se spune ca are aspect de "vale" daca elementele descresc pana la un moment dat, apoi cresc. De ex. 10 8 6 9 11 13).

```

domains
  el = integer
  list = el*

```

predicates

```

%pct b
suma(list,integer) %(i,o)
sumaAlt(list,integer,integer) %(i,i,o)

```

clauses

```

%pct a
vale([],2,_).
vale([H|[]],2,_).
vale([H|[H1|T]],Pas,Nr):-Pas = 1,
                          H>H1,!,
                          L=[H1|T],
                          Nr1=Nr+1,
                          vale(L,Pas,Nr1).

```

```

vale([H|[H1|T]],Pas,Nr):-Pas = 1,
                          H<H1,
                          Nr>1,!,
                          L=[H1|T],
                          Pas1=Pas + 1,
                          Nr1=Nr+1,
                          vale(L,Pas1,Nr1).

```

```

vale([H|[H1|T]],Pas,Nr):-Pas = 2,
                          H<H1,!,
                          L=[H1|T],
                          Nr1=Nr+1,
                          vale(L,Pas,Nr1).

```

```

rezolvare(L1):-vale(L1,1,1).

```

- b.** Sa se calculeze suma alternanta a elementelor unei liste
($I_1 - I_2 + I_3 \dots$).

16.

- a.** Sa se scrie un predicat care elimina dintr-o lista toate elementele care apar o singura data (ex: $I=[1,2,1,4,1,3,4] \Rightarrow I=[1,1,4,4]$)
- b.** Sa se scrie un predicat care sa testeze daca o lista formata din numere intregi are aspect de "munte"(o multime se spune ca are aspect de "munte" daca elementele cresc pana la un moment dat, apoi descresc. De ex. 1 6 8 9 7 2).

domains

el = integer
list = el*

predicates

%pct a
member(el,list)
eliminNr(list,list) %(i,o)

%pct b
munte(list,integer,integer)
rezolvare(list)

clauses

%pct a
member(E,[E|_]).
member(E,[_|T]):-member(E,T).

eliminNr([],[]).
eliminNr([H|T],[H]):-T=[],!.
eliminNr([H|T],L):-not(member(H,T)),!,
 eliminNr(T,L).
eliminNr([H|T],[H|L]):-eliminNr(T,L).

%pct b
munte([],2,_).
munte([H|[]],2,_).
munte([H|[H1|T]],Pas,Nr):-Pas = 1,
 H<H1,!,
 L=[H1|T],
 Nr1=Nr+1,
 munte(L,Pas,Nr1).

munte([H|[H1|T]],Pas,Nr):-Pas = 1,
 H>H1,
 Nr>1,!,
 L=[H1|T],
 Pas1=Pas + 1,

```

        Nr1=Nr+1,
        munte(L,Pas1,Nr1).
munte([H|[H1|T]],Pas,Nr):-Pas = 2,
        H>H1,!,
        L=[H1|T],
        Nr1=Nr+1,
        munte(L,Pas,Nr1).

rezolvare(L1):-munte(L1,1,1).

```

P2: Liste in Prolog (2)

1. Definiti un predicat care determina succesorul unui numar reprezentat cifra cu cifra intr-o lista. De ex: [1 9 3 5 9 9] --> [1 9 3 6 0 0]

```

domains
    el = integer
    list = el*

```

predicates

```

adaugSf(el,list,list)  %(i,i,o)
invers(list,list)      %(i,o)
sum(list,integer,list) %(i,i,o)
succesor(list,list)    %(i,o)
tipar(list)

```

clauses

```

adaugSf(E,[],[E]).
adaugSf(E,[H|T],[H|L]):-adaugSf(E,T,L).

```

```

invers([],[]).
invers([H|T],L):-invers(T,L1),
    adaugSf(H,L1,L).

```

```

succesor(L,LS):- invers(L,LI),
    sum(LI,1,LA),
    invers(LA,LS),
    write("L="),
    tipar(LS),
    write("]").

```

```

sum([],Tr,[Tr]):-Tr<>0,!.

```

```

sum([],Tr,[]):-!.
sum([H|T],Tr,[S|L1]):-
    S1=H+Tr,
    S=S1 mod 10,
    Tr1=S1 div 10,
    sum(T,Tr1,L1).

```

```

tipar([]).
tipar([H|T]):-write(H),
    write(","),
    tipar(T).

```

2. Definiti un predicat care determina predecesorul unui numar reprezentat cifra cu cifra intr-o lista. De ex: [1 9 3 6 0 0] --> [1 9 3 5 9 9]

```

%!!!!!!Nu merge pt 10->9,in rest merge
domains
    el = integer
    list = el*

```

predicates

```

adaugSf(el,list,list)  %(i,i,o)
invers(list,list)      %(i,o)
sum(list,integer,list) %(i,i,o)
predecesor(list,list)  %(i,o)
tipar(list)

```

clauses

```

adaugSf(E,[],[E]).
adaugSf(E,[H|T],[H|L]):-adaugSf(E,T,L).

```

```

invers([],[]).
invers([H|T],L):-invers(T,L1),
    adaugSf(H,L1,L).

```

```

predecesor(L,LS):- invers(L,LI),
    sum(LI,-1,LA),
    invers(LA,LS),
    write("L="),

    tipar(LS),
    write(" ").

```

```

sum([],Tr,[]):-!.
sum([H|T],Tr,[S|L1]):-H=0,
    S=9,

```



```

        Tr1=-1,
        sum(T,Tr1,L1).
sum([H|T],Tr,[S|L1]):-H<>0,
    S=H+Tr,
    Tr1=0,
    sum(T,Tr1,L1).

tipar([]).
tipar([H|T]):-write(H),
    write(","),
    tipar(T).

```

3. Definiti un predicat care determina suma a doua numere scrise in reprezentare de lista.

```

domains
    el = integer
    list = el*

predicates
    length(list,integer)
    zero(integer,list,list)
    maxim(integer,integer,integer)
    adun(list,list,list,integer)
    suma(list,list,list)
clauses

length([],0):-!.
length([H|T],N):-length(T,N1),
    N=N1+1.

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

zero(0,X,X):-!.
zero(N,X,[0|X1]):-N1=N-1,
    zero(N1,X,X1).

adun([],[],[],0):-!.
adun([A|L1],[B|L2],[Cifra|L3],Tr):-
    adun(L1,L2,L3,Tr2),
    Cifra=(A+B+Tr2)mod 10,
    Tr=(A+B+Tr2)div 10.

suma(A,B,Rez):-
    length(A,LA),
    length(B,LB),
    maxim(LA,LB,LMax),
    C1=LMax-LA,

```

```

C2=LMax-LB,
zero(C1,A,AA),
zero(C2,B,BB),
adun(AA,BB,Rez,0).

```

4. Definiti un predicat care determina diferenta a doua numere scrise in reprezentare de lista.
5. Definiti un predicat care determina valoarea unui polinom intr-un punct. Polinomul se da sub forma listei coeficientilor sai.

```

domains
  el=integer
  list=el*
predicates
  polin(list,el,el,el)
  polinom(list,el,el)
clauses
  polin([H],X,S1,S2):- S2=S1*X+H,!.
  polin([H|T],X,S1,S2):- S3=S1*X+H,polin(T,X,S3,S2).
  polinom(L,X,S):-polin(L,X,0,S).

```

6. Definiti predicatele de egalitate si mai mic pentru numere scrise in reprezentate pe liste.

```

domains

  el=integer
  lista=el*

predicates

  lung(lista,el)
  compara(lista,lista,integer)
  tip(lista,lista)

clauses

  lung([],0).
  lung([_|T],L):-lung(T,rez),L=rez+1.

```

```

compara([],[],0).
compara(L1,L2,rez):-lung(L1,X),lung(L2,Y),X<Y,!,rez=-1.
compara(L1,L2,rez):-lung(L1,X),lung(L2,Y),X>Y,!,rez=1.
compara([H1|_],[H2|_],rez):-H1<H2,!,rez=-1.
compara([H1|_],[H2|_],rez):-H1>H2,!,rez=1.
compara([H1|T1],[H2|T2],rez):-H1=H2,compara(T1,T2,rez).

tip(L1,L2):-compara(L1,L2,1),write("Primul sir este mai mare."),nl.
tip(L1,L2):-compara(L1,L2,0),write("Egale"),nl.
tip(L1,L2):-compara(L1,L2,-1),write("Al doilea sir este mai mare."),nl.

```

7. Sa se sorteze o lista cu pastrarea dublurilor.

Exemplu: [4 2 6 2 3 4] --> [2 2 3 4 4 6]

```

domains
    el = integer
    list = el*

predicates
    inserare(el,list,list)  %(i,i,o)
    sort(list,list)  %(i,o)

clauses
    inserare(E,[],[E]):-!.
    inserare(E,[H|T],[H|L]):-E>=H,!,
                                inserare(E,T,L).
    inserare(E,[H|T],[E|[H|T]]):-!.

    sort([],[]):-!.
    sort([H|T],L):-sort(T,L1),
                    inserare(H,L1,L).

```

8. Sa se sorteze o lista cu eliminarea dublurilor.

Exemplu: [4 2 6 2 3 4] --> [2 3 4 6]

```

domains
    el = integer
    list = el*

predicates
    elimin(el,list,list)  %(i,i,o)
    inserare(el,list,list)  %(i,i,o)

```

```

    sort(list,list)      %(i,o)

clauses

    elimin(_,[],[]).
    elimin(E,[E|T],L):-!,elimin(E,T,L).
    elimin(E,[H|T],[H|L]):-elimin(E,T,L).

    inserare(E,[],[E]):-!.
    inserare(E,[H|T],[H|L]):-E>=H,!,
                                inserare(E,T,L).
    inserare(E,[H|T],[E|[H|T]]):-!.

    sort([],[]):-!.
    sort([H|T],L):-elimin(H,T,L1),
                    sort(L1,L2),
                    inserare(H,L2,L).

```

9. Sa se interclaseze cu pastrarea dublurilor doua liste sortate.

```

domains
    el = integer
    list = el*

predicates
    inter(list,list,list)  %(i,o)

clauses
    inter([],[],[]).
    inter([],L2,L2):-!.
    inter(L1,[],L1):-!.
    inter([H1|L1],[H2|L2],[H1|L3]):-H1<H2,!,
                                    L=[H2|L2],
                                    inter(L1,L,L3).
    inter([H1|L1],[H2|L2],[H2|L3]):-H1>=H2,!,
                                    L=[H1|L1],
                                    inter(L,L2,L3).

```

10. Sa se interclaseze fara pastrarea dublurilor doua liste sortate.

```

*/
%!!!!!!Nu-i chiar corecta!!!!!!!!!!!!

```


- 12.** Intr-o lista L sa se inlocuiasca toate aparitiile unui element E cu elementele unei alte liste, L1. Exemplu: `inloc([1,2,1,3,1,4],1,[10,11],X)` va produce `X=[10,11,2,10,11,3,10,11,4]`.

```
domains
  el = integer
  list = el*
```

```
predicates
  concatenare(list,list,list)  %(i,i,o)
  subst(el,list,list,list)  %(i,i,i,o)
```

```
clauses
  concatenare([],L,L).
  concatenare([H|L1],L2,[H|L3]):-concatenare(L1,L2,L3).

  subst(_,_,[],[]).
  subst(E,L,[E|T],R):-subst(E,L,T,R1),!,
                        concatenare(L,R1,R).
  subst(E,L,[H|T],[H|R]):-subst(E,L,T,R).
```

- 13.** Definiti un predicat care determina produsul unui numar reprezentat cifra cu cifra intr-o lista cu o anumita cifra. De ex: `[1 9 3 5 9 9] * 2`
`[3 8 7 1 9 8]`

```
domains
```

```
list=integer*
```

```
predicates
```

```
prod(list,integer,integer,list)    %(i,i,i,o)
produs(list,integer,list)          %(i,i,o)
add_last(integer,list,list)        %(i,i,o)
reverse(list,list)                  %(i,o)
```

```
clauses
```

```
add_last(E,[],[E]).
add_last(E,[H|T],[H|L]):-add_last(E,T,L).

reverse([],[]).
reverse([H|T],L):-reverse(T,L1),!,
                  add_last(H,L1,L).
```

```

prod([],_,TR,[]):-TR=0.
prod([],_,TR,[TR]):-TR>0.
prod([H|T],C,TR,[H1|RR]):-
    H1=((H*C) mod 10)+TR,
    T1=(H*C) div 10,
    prod(T,C,T1,RR).

produs(N,C,R):-reverse(N,NR),
    prod(NR,C,0,RR),
    reverse(RR,R).

```

14. Sa se determine pozitiile elementului minim dintr-o lista liniara. De ex:
 poz([10,-14,12,13,-14], L) va produce L = [2,5].

```

domains
    el = integer
    list = el*

```

```

predicates
    minim(list,el)  %(i,o)
    detPoz(el,el,list,list)  %(i,i,o)
    pozitii(list,list)  %(i,o)

```

clauses

```

minim([],3200):-!.
minim([H|T],M):-minim(T,M),M<H,!.
minim([H|_],H):-!.

detPoz(_,_,[],[]).
detPoz(E,N,[E|T],[N|L]):-!,N1=N+1,
    detPoz(E,N1,T,L).

detPoz(E,N,[H|T],L):-N1=N+1,
    detPoz(E,N1,T,L).

pozitii(L,L1):-minim(L,M),
    detPoz(M,1,L,L1).

```

15. Dandu-se o lista liniara numerica, sa se stearga toate secventele de valori

consecutive. Ex: `sterg([1, 2, 4, 6, 7, 8, 10], L)` va produce `L=[4, 10]`.

domains

`el = integer`

`list = el*`

predicates

`elimin(list,list)`

clauses

`elimin([],[]).`

`elimin([H1|[H2|[H3|T]]],L):-H3=H2+1,
H2=H1+1,!,
L1=[H2|[H3|T]],
elimin(L1,L).`

`elimin([H1|[H2|[H3|T]]],L):-H2=H1+1,
H3<>H2+1,
!,
L1=[H3|T],
elimin(L1,L).`

`elimin([H1|[H2|[]]],[]):-H2=H1+1,!.
elimin([H|T],[H|L]):-elimin(T,L).`

16. Dandu-se o lista liniara numerica, sa se stearga toate secventele de valori crescatoare. Ex: `sterg([1, 2, 4, 6, 5, 7, 8, 2, 1], L)` va produce `L=[2, 1]`.

domains

`el = integer`

`list = el*`

predicates

`elimin(list,list) %(i,o)`

clauses

`elimin([],[]).`

`elimin([H1|[H2|[H3|T]]],L):-H1<H2,
H2<H3,!,
L1=[H2|[H3|T]],
elimin(L1,L).`

`elimin([H1|[H2|[H3|T]]],L):-H1<H2,
H2>=H3,!,
L1=[H3|T],
elimin(L1,L).`

`elimin([H1|[H2|[]]],[]):-H1<H2,!.
elimin([H|T],[H|L]):-elimin(T,L).`

$\text{elimin}([H|T],[H|L]):-\text{elimin}(T,L).$