1. Sa se scrie un predicat care intoarce lungimea unei liste de simboluri.

```
domains
list=symbol*
predicates
lungime(list,integer)
clauses
lungime([],0).
lungime([_|T],Coada):-lungime(T,X),Coada=X+1.
```

2. Sa se scrie un predicat care intoarce suma elementelor dintr-o lista.

```
domains
lista=integer*

predicates
suma(lista,integer)

clauses
suma([],0).
suma([H|T],TS):-suma(T,S),TS=H+S .
```

3. Sa se scrie un predicat care determina toate aparitiile unui element intr-o lista.

```
domains
element=integer
lista=element*
predicates
nraparitie(element,lista,integer)
clauses
nraparitie(_,[],0):-!.
nraparitie(E,[E|T],A):-!, nraparitie(E,T,Ap),A=Ap+1.
nraparitie(E,[_|T],A):- nraparitie(E,T,A).
```

4. Sa se scrie un predicat care pune cap la cap doua liste.

```
domains
element=integer
lista=element*
predicates
unire(lista,lista,lista)
clauses
unire([],Lista,Lista).
unire([H|T],Lista2,[H|Rezultat]):-unire(T,Lista2,Rezultat).
```

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

```
domains
element=integer
lista=element*

predicates
lungime(lista,element)
set(lista,lista)
multime(lista)
membru(element,lista)
clauses
lungime([],0).
lungime([_|T],L):-lungime(T,T2),L=T2+1.
membru(E,[E|_]).
membru(E,[_|T]):-membru(E,T).
set([],[]).
set([H|T],[H|Out]):-not(membru(H,T)),set(T,Out).
set([H|T],Out):-membru(H,T),set(T,Out).
multime(Lista):-set(Lista,L1),lungime(Lista,Lung1),lungime(L1,Lung1).
```

6. Sa se scrie un predicat care transforma o lista intr-o multime.

```
domains
element=integer
lista=element*

predicates
lungime(lista,element)
set(lista,lista)
membru(element,lista)
clauses
lungime([],0).
lungime([_|T],L):-lungime(T,T2),L=T2+1.
membru(E,[E|_]).
membru(E,[_|T]):-membru(E,T).
set([],[]).
set([H|T],[H|Out]):-not(membru(H,T)),set(T,Out).
set([H|T],Out):-membru(H,T),set(T,Out).
```

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

```
domains
element=integer
lista=element*

predicates
unire(lista,lista,lista)
membru(element,lista)
clauses
membru(E,[E|_]).
membru(E,[_|T]):-membru(E,T).
unire([],X,X).
unire([X|R],Y,Z):-membru(X,Y),!,unire(R,Y,Z).
unire([X|R],Y,[X|Z]):-unire(R,Y,Z).
```

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

```
domains
element=integer
lista=element*

predicates
intersectie(lista,lista,lista)
membru(element,lista)
clauses
membru(E,[E|_]).
membru(E,[_|T]):-membru(E,T).
intersectie([],_,[]).
intersectie([X|R],Y,[X|Z]):-membru(X,Y),!,intersectie(R,Y,Z).
intersectie([_|R],Y,Z):-intersectie(R,Y,Z).
```

9. Sa se scrie un predicat care intoarce diferenta a doua multimi.

```
domains
element=integer
lista=element*
predicates
membru(element,lista)
dif(lista,lista,lista)
clauses
membru(E,[E|_]).
membru(E,[_|T]):- membru(E,T).
dif([],_,[]):-!.
dif([H|T],B,C):-membru(H,B),!,dif(T,B,C).
dif([H|T],B,[H|T2]):-dif(T,B,T2).
```

10. Sa se scrie un predicat care testeaza egalitatea a doua multimi, fara sa se faca apel la diferenta a doua multimi.

```
domains
element=integer
lista=element*

predicates
membru(element,lista)
egalitate(lista,lista)

clauses
membru(E,[E|_]).
membru(E,[_|T]):-membru(E,T).
egalitate([H|T],[H2|T2]):-H=H2;membru(H,T2),egalitate(T,T2).
```

11. Sa se scrie un predicat care sterge toate aparitiile unui atom dintr-o lista.

```
domains
element=integer
lista=element*
predicates
sterge(element,lista,lista)
clauses
```

```
sterge(_,[],[]).
sterge(Atom,[Atom|T],Tr):-sterge(Atom,T,Tr).
sterge(Atom,[H|T],[H|T2]):-not(H=Atom), sterge(Atom,T,T2).
```

12. 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.

```
domains
element=integer
lista=element*
predicates
nrpar_elem(lista)

clauses
nrpar_elem([_,_]).
nrpar_elem([_,_|Rest]):-nrpar_elem(Rest).
```

13. Sa se scrie un predicat care substituie intr-o lista un element prin altul.

```
domains
element=integer
lista=element*
predicates
inlocuire(element,element,lista,lista)
clauses
inlocuire(_,_,[],[]).
inlocuire(Atom,Element,[Atom|T1],[Element|T2]):-inlocuire(Atom,Element,T1,T2).
inlocuire(Atom,Element,[H|T1],[H|T2]):-not(H=Atom), inlocuire(Atom,Element,T1,T2).
```

14. Definiti un predicat care inverseaza o lista.

```
domains
element=integer
lista=element*

predicates
inverseaza(lista,lista)
adauga(lista,lista,lista)
clauses
adauga([],R,R).
adauga([H|T],R,[H|X]):-adauga(T,R,X).
inverseaza([X],[X]).
inverseaza([H|T],R):-inverseaza(T,X),adauga(X,[H],R).
```

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

```
domains
element=integer
lista=element*
predicates
sterge(element,lista,element,lista)

clauses
sterge(X,[X|T1],1,T1).
sterge(X,[H|T1],N,[H|T2]):-N>1,N1=N-1,sterge(X,T1,N1,T2).
```

16. Sa se elimine elementul de pe pozitia a n-a a unei liste liniare.

```
domains
element=integer
lista=element*
predicates
sterge(element,lista,element,lista)

clauses
sterge(X,[X|T1],1,T1).
sterge(X,[H|T1],N,[H|T2]):-N>1,N1=N-1,sterge(X,T1,N1,T2).
```

17. 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
nr_apar(integer,lista,integer)
get_perechi(lista,perechi)
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).
```

18. Definiti un predicat care selecteaza al n-lea element al unei liste.

```
domains
element=integer
lista=element*
predicates
select(element,lista,element)

clauses
select(X,[X|T1],1).
select(X,[H|T1],N):-N>1,N1=N-1,select(X,T1,N1).
```

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

```
domains
list=integer*
predicates
cmmdc(integer,integer,integer
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).
```

20 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 dl).

```
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).
```

23. 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
   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).
```

24. elementelor pare si impare.
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)
invers(list,list)
sum(list,integer,list)
succesor(list,list)
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)
```

25. 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]

```
domains
el = integer
  list = el*
predicates
  adaugSf(el,list,list)
invers(list,list
sum(list,integer,list)
predecesor(list,list
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).
```

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

```
domains
  el = integer   1
ist = 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).
```

29. 29. Oefiniti predicatele de egalitate si rnai rnic pentru nurnere 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.
```