%1.aSa 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.

%a.

%member(E:Integer,L:List).

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

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

%diferenta(L1:List,L2:List, LR:List).

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).

%b.

%adauga(L:List, LR:List).

adauga([],[]).

adauga([H|T],[H|[1|T1]]):-

H mod 2 =:= 0,

!,

adauga(T,T1).

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

%2.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.

%cmmdc(E1:Integer, E2:Integer, R:Integer).

cmmdc(A,B,A):- A=B,!.

cmmdc(A,B,D):- A<B ,!, B1 is B-A , cmmdc(A,B1,D).

cmmdc(A,B,D):- A>B ,!,A1 is A-B, cmmdc(A1,B,D).

%cmmmc(E:Integer, F:Integer, R:Integer).

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

%cmmmcList(L:List, E:Integer).

cmmmcList([],1).

cmmmcList([H|T],R):-cmmmcList(T,S),cmmmc(H,S,R).

%b

%adaug\_aux(L:List, V:Integer, P:Integer, C:Integer, LR:List).

adaug\_aux([],\_,\_,\_,[]).

adaug\_aux([H|T],V,P,C,[H|[V|L]]):-

P=C,

!,

P2 is P\*2,

C2 is C+1,

adaug\_aux(T,V,P2,C2,L).

adaug\_aux([H|T],V,P,C,[H|L]):-

not(P = C),

!,

C2 is C+1,

adaug\_aux(T,V,P,C2,L).

adaug(L,V,R):-

P = 1,

C = 1,

adaug\_aux(L,V,P,C,R).

%3.aSa se scrie un predicat care transforma o listaintr-o multime, in

% ordinea primei aparitii. Exemplu: [1,2,3,1,2] e transformat in [1,2,3].

%sterge(E:Intreg, L:Lista, LR:Lista);

%E:elementul care se doreste sa fie sters

%L:lista intiala, din care se va sterge elementul;

%LR:lista rezultata dupa stergerea elementului din lista initiala;

%Model de flux: (i,i,o);

sterge(\_,[],[]):-!.

sterge(E,[E],[]):-!.

sterge(E,[H|T],[H|R]):- H=\=E, sterge(E,T,R).

sterge(E,[H|T],R):-H=:=E, sterge(E,T,R).

%multime(L:Lista, LR:Lista);

%L:lista initiala ce se doreste a fi modificata in multime;

%LR:lista rezultat, sub forma de multime,

%in functie de prima aparitie a elementelor;

%Model de flux: (i,o), (i,i);

multime([],[]):-!.

multime([H|T],[H|R]):-sterge(H,T,L), multime(L,R).

%3.bSa 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.

%liste(L:Lista, P:Lista, I:Lista);

%Lista = Element\*;

%Model de flux: (i,o,o), (i,i,i);

%L:lista care se doreste separata;

%P:lista formata din elemente pare din lista initiala;

%I:lista formata din elemente impare din lista initiala;

%

%Implementare PROLOG:

liste([],[],[]):-!.

liste([H|T],[H|P],I):- H1 is H mod 2, H1=:=0, liste(T,P,I), !.

liste([H|T], P, [H|I]):- H1 is H mod 2, H1=\=0, liste(T,P,I), !.

%lungime(L:Lista, N:Intreg);

%Lista = Element\*;

%Model de fux: (i,o), (i,i);

%L:lista a carui lungime se doreste a fi calculata;

%N:lungimea listei;

lungime([],0):-!.

lungime([\_|T], N1):- lungime(T,N), N1 is N+1,!.

%pare\_impare(L:Lista, [P:Lista, I:Lista], NP:Intreg, NI:Intreg);

%model de flux (i,[o,o], o,o)

%L:lista intiala;

%P:lista de numere pare din lista initiala;

%I:lista de numere impare din lista initiala;

%NP:numarul elementelor pare din lista initiala;

%NI:numarul de elemente impare din lista initiala;

pare\_impare(L, [P,I],NP,NI):- liste(L,P,I), lungime(P,NP), lungime(I,NI).

%4.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.

%concatenare(L1:List,L2:List,LR:List).

concatenare([],L,L).

concatenare([H|T],L1,[H|R]):-concatenare(T,L1,R).

%subst(E:Integer, L1:List, L2:List,LR:List).

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).

%b

%elimina(L:List, E:Integer, R:List).

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

elimina([\_|T],1,T):-!.

elimina([H|T],N,[H|L]):- N1 is N-1, elimina(T,N1,L).

%5a. 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]].

%a.

%sterge(E:Integer, L:List, LR:List).

sterge(\_, [],[]):-!.

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

H \= E,

!,

sterge(E,T,T1).

sterge(E, [\_|T],T1):-

sterge(E,T,T1).

%b.

%nr\_apar(E:integer, L:List, N:Integer).

nr\_apar(\_, [], 0).

nr\_apar(E, [H|T],N):- H=E,nr\_apar(E,T,NT),N is NT + 1,!.

nr\_apar(E, [H|T], N):- H\= E,nr\_apar(E,T,N).

%perechi(L:List, P:List).

perechi([], []):-!.

perechi([E],[[E,1]]):-!.

perechi([H|T],[[H,N1]|P]):-

nr\_apar(H,T,N),

N1 is N+1,

sterge(H,[H|T],L),

!,

perechi(L,P).

%6.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] => l=[2,3])

% b. Sa se elimine toate aparitiile elementului maxim dintr-o lista de

% numere intregi.

%a.

%member(E:Integer, L:List).

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

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

%elimin(E:Integer, L:List, LR:List).

elimin(\_,[],[]):-!.

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

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

%elimina(L:List, LR:List).

elimina([],[]):-!.

elimina([H|T],L):-

member(H,T),!,

elimin(H,T,L1),

elimina(L1,L).

elimina([H|T],[H|L]):-

not(member(H,T)),

!,

elimina(T,L).

%b.

%maxim(L:List, E:Integer).

maxim([], -3200):-!.

maxim([H|T],H):-maxim(T,M),H>M,!.

maxim([\_|T],M):-maxim(T,M).

%eliminaMax(L:List, LR:List).

eliminaMax(L,LR):-maxim(L,M), elimin(M,L,LR).

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

%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]].

%a.

%apartine(E:Integer, L:List).

apartine(E,[E|\_]):-!.

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

%reuniune(L1:List, L2:List, LR:List).

reuniune([],T1,T1).

reuniune([H|T],L2,[H|T1]):-

not(apartine(H,L2)),

!,

reuniune(T,L2,T1).

reuniune([\_|T],L2,T1):- reuniune(T,L2,T1).

%b.

%pereche(E:Integer, L:List, LR:List).

pereche(E,[H|\_],[E,H]):- E\=H.

pereche(E,[\_|T],P):-pereche(E,T,P).

%perechi(L:List, LR:List).

perechi([H|T],P):-pereche(H,T,P).

perechi([\_|T],P):-perechi(T,P).

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

% b. Sa se construiasca lista (m, ..., n), adica multimea numerelor

% intregi din intervalul [m, n].

%

%apartine(E:Integer, L:List).

apartine(E,[E|\_]):-!.

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

%inters(L1:List, L2:List, LR:List).

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

inters([H|T],L2,[H|LR]):-apartine(H,L2),!, inters(T,L2,LR).

inters([H|T],L2,LR):-not(apartine(H,L2)),inters(T,L2,LR).

%b.

%interval(M:Integer,N:Integer, L:List).

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 is M+1, interval(M2,N,T).

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

%b. Definiti un predicat care intoarce cel mai mare divizor comun al

%numerelor dintr-o lista.

%a.

%verif(L:List, P:Integer, N:Integer, LR:List).

insert([],0,E,[E]):-!.

insert(L,P,E,LR):-

P=0,!,

LR = [E|L].

insert([H|T],P,E,LR):-

N is P-1,

insert(T,N,E,LN),

LR = [H|LN].

%b.

%cmmdc(A:Integer, B:Integer).

cmmdc(A,B,A):-A=B,!.

cmmdc(A,B,D):-A<B,!,B1 is B-A, cmmdc(A,B1,D).

cmmdc(A,B,D):-A>B,!, A1 is A-B,cmmdc(A1,B,D).

%cmmdcL(L:List, R:Integer).

cmmdcL([H],H):-!.

cmmdcL([H1|[H2|T]],R):-

cmmdc(H1,H2,R1),

cmmdcL([R1|T],R).