```
% BAZA DE CUNOSTINTE
/* Clauze:
fapte.
        reguli:
fapt :- succesiune de fapte.
*/

/*
?- V=2025**150.
?- =(V,2025**150).
     **
V=  /  \
  2025 150
?- V is 2025**150.
?- is(V,2025**150).
?- is(V,**(2025,150)).
?- tell('c:/tempwork/calcul.txt'),V is 2025**150,write(V),told.
*/

/* Valorile booleene: date de constantele: false si true.
Conectori logici predefiniti in Prolog:
conjunctia (si):   ,
disjunctia (sau):  ;
negatia:        not   \+
Prologul lucreaza cu predecente, care sunt opusul prioritatilor (i.e. cu cat precedenta unui termen
(= precedenta operatorului sau dominant, in cazul unui termen compus) e mai mare, cu cat prioritatea
sa e mai mica; si, desigur, vice-versa) si au ca valori numere naturale:
constantele, termenii inclusi intre paranteze (rotunde) si termenii de forma
numeOperator(listaArgumenteSeparatePrinVirgula) au precedenta 0, deci prioritatea cea mai mare;
conjunctia are precedenta mai mica, deci prioritatea mai mare decat a disjunctiei.
?- false,false.
```

```
?- false,true.
?- true,false.
?- true,true.
?- false;false.
?- false;true.
?- true;false.
?- true;true.
?- not(false).
?- not(true).
?- \+(false).
?- \+(true).
*/

/*
?- 10*5 = 20+30.
  *      +
 / \   = / \
10  5   20 30
?- not(10*5 = 20+30).
?- 10*5 \= 20+30.
?- 10*5 =:= 20+30.
?- not(10*5 =:= 20+30).
?- 10*5 =\= 20+30.
?- 10*5 > 20+30.
?- 10*5 < 20+30.
?- 10*5 >= 20+30.
?- 10*5 =< 20+30.
?- V=X.
?- Variabila=_totVariabila.
?- V=20.
?- V=constanta.
```

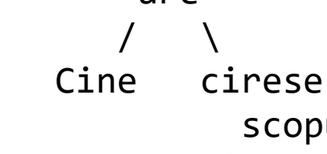
```
?- V=10+20.
?- V=f(a,1,b,g(X),c,Y).
?- V=f(V).
     f
V =  |
     f
     |
     f
     |
     f
     |
     V
    ...
?- unify_with_occurs_check(V,f(V)).
?- V == W.
?- V \== W.
?- not(V == W).
*/

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% fapte:

prop.
altaprop.

are(ana,mere).
are(maria,pere).
are(vali,mere).
are(_,cirese).

/* reguli: cu operatorul dominant dat de predicatul neck, semnificand "daca":  :-
```

```
fapt :- succesiune de fapte.
:-(fapt,succesiune de fapte).
Faptele din argumentul drept al predicatului neck pot fi unice sau legate prin conectori logici.
*/

are(Cineva,gutui) :- are(Cineva,mere).

are(nicu,Fructe) :- are(ana,Fructe), are(maria,Fructe). % echivalent:
:-(are(nicu2,Fructe), (are(ana,Fructe), are(maria,Fructe))).

are(bodgan,Fructe) :- are(ana,Fructe); are(maria,Fructe).   % echivalent:
are(bodgan2,Fructe) :- are(ana,Fructe).
are(bodgan2,Fructe) :- are(maria,Fructe).

are(alex,Fructe) :- are(ana,Fructe) ; are(maria,Fructe), are(vali,Fructe). % echivalent:
are(alex2,Fructe) :- are(maria,Fructe), are(vali,Fructe).

are(marius,Fructe) :- are(ana,Fructe) , (are(maria,Fructe); are(vali,Fructe)).
        % echivalent:
are(marius2,Fructe) :- are(ana,Fructe) , are(maria,Fructe).
are(marius2,Fructe) :- are(ana,Fructe) , are(vali,Fructe).
        % pentru ca regula pentru marius de mai sus e echivalenta cu:
are(marius3,Fructe) :- are(ana,Fructe), are(maria,Fructe) ; are(ana,Fructe), are(vali,Fructe).
        % in conformitate cu distributivitatea de mai jos,
   % asadar aceasta regula e echivalenta cu cele doua reguli de mai sus pentru marius2

/* Conjunctia e distributiva fata de disjunctie:
[p si (q sau r)] <=> [(p si q) sau (p si r)]   */

/* La interogarile:
```

```
?- prop.
?- altaprop.
?- prop, altaprop.
Prolog-ul raspunde true.
Dati si interogarile:
?- prop; altaprop.
?- not(prop); altaprop.
?- prop; not(altaprop).
?- not(prop); not(altaprop).
?- not(prop; altaprop).
Apoi dati interogarile:
?- are(ana,mere).
?- are(ana,pere).
?- are(ana,cirese).
?- are(ana,gutui).
?- are(ana,Ce).
Sa dam interogarea:
?- are(Cine,cirese).
Prolog-ul va incerca sa unifice scopul are(Cine,cirese) cu faptele si membrii stangi ai regulilor
din baza de cunostinte.
     are                                  are
    /   \       \= prop, altaprop,      /   \    , are(maria,pere), are(vali,mere)
 Cine   cirese                        ana   mere
     are             are
    /   \       =   /   \          => true: PRIMA SOLUTIE, fiind rezultatul unificarii
 Cine   cirese     V    cirese        scopului are(Cine,cirese) cu faptul are(_,cirese),
                                      care semnifica: are(_,cirese) e adevarat
                \= are(Cineva,gutui)
Avem aceasta unificare a scopului are(Cine,cirese) cu membrul stang are(nicu,Fructe) al regulii:
are(nicu,Fructe) :- are(ana,Fructe), are(maria,Fructe).
     are             are
```

```
    /   \       =   /   \     <=> Cine=nicu, Fructe=cirese => urmatorul scop este membrul
 Cine   cirese    nicu  Fructe    drept al aceleiasi reguli cu variabilele inlocuite cu
                                  valorile lor rezultate din aceasta unificare; asadar
avem urmatorul scop intermediar: ?- are(ana,cirese), are(maria,cirese).
     are                              are
    /   \       \= prop, altaprop,   /   \    , are(maria,pere), are(vali,mere)
  ana   cirese                     ana   mere
     are            are
    /   \       =   /   \
  ana   cirese     V    cirese
                \= are(Cineva,gutui), are(nicu,Fructe), are(nicu2,Fructe),
                \= are(bodgan,Fructe), are(bodgan2,Fructe)
                \= are(alex,Fructe), are(alex2,Fructe)
                \= are(marius,Fructe), are(marius2,Fructe), are(marius3,Fructe)
     are            are
    /   \       =   /   \       => true pentru membrul stang al scopului intermediar
  ana   cirese     V    cirese     compus de mai sus
     are                              are
    /   \       \= prop, altaprop,   /   \    , are(maria,pere), are(vali,mere)
maria   cirese                     ana   mere
     are            are
    /   \       =   /   \       => true pentru membrul drept al scopului intermediar
maria   cirese     V    cirese     compus are(ana,cirese), are(maria,cirese)
=> true pentru scopul intermediar: are(ana,cirese), are(maria,cirese)
=> Cine=nicu: A DOUA SOLUTIE pentru scopul initial are(Cine,cirese) (partea Fructe=cirese din
unificarea Cine=nicu, Fructe=cirese nu apare in solutie, pentru ca variabila Fructe nu apare in
scopul are(Cine,cirese)).
are(maria,cirese)\= are(Cineva,gutui), are(nicu,Fructe), are(nicu2,Fructe),
                \= are(bodgan,Fructe), are(bodgan2,Fructe)
                \= are(alex,Fructe), are(alex2,Fructe)
                \= are(marius,Fructe), are(marius2,Fructe), are(marius3,Fructe)
```

=> Nu exista alte solutii pentru scopul intermediar are(ana,cirese), are(maria,cirese).



<=> Cine=nicu2, Fructe=cirese => la fel ca mai sus, va rezulta solutia Cine=nicu2: A TREIA SOLUTIE pentru scopul initial are(Cine,cirese).

In aceeasi maniera, din unificarile:

are(Cine,cirese) = are(bodgan,Fructe) => Cine=bodgan: A PATRA SOLUTIE pentru scopul are(Cine,cirese)

are(Cine,cirese) = are(bodgan2,Fructe) => Cine=bodgan2: A CINCIA SOLUTIE pentru scopul are(Cine,cirese)

are(Cine,cirese) = are(alex,Fructe) => Cine=alex: A SASEA SOLUTIE pentru scopul are(Cine,cirese)

are(Cine,cirese) = are(alex2,Fructe) => Cine=alex2: A SAPTEA SOLUTIE pentru scopul are(Cine,cirese)

are(Cine,cirese) = are(marius,Fructe) => Cine=marius: A OPTA SOLUTIE pentru scopul are(Cine,cirese)

are(Cine,cirese) = are(marius2,Fructe) => Cine=marius2: A NOUA SOLUTIE pentru scopul are(Cine,cirese)

are(Cine,cirese) = are(marius3,Fructe) => Cine=marius3: A ZECEA si ULTIMA SOLUTIE pentru scopul are(Cine,cirese)

Dati si interogarile:

```
?- are(Cine,Ce).
?- setof((Cine,Ce), are(Cine,Ce), L), write(L).
?- setof(are(Cine,Ce), are(Cine,Ce), L), write(L).
```

Metapredicate predefinite:

setof(Termen,Scop,Lista)=true <=> Lista este lista fara duplicate a termenilor de forma Termen care satisfac conditia Scop, in cazul in care exista astfel de termeni, si intoarce false in caz contrar;

bagof(Termen,Scop,Lista)=true <=> Lista este lista cu duplicate a termenilor de forma Termen care satisfac conditia Scop, in cazul in care exista astfel de termeni, si intoarce false in caz contrar;

findall(Termen,Scop,Lista)=true <=> Lista este lista cu duplicate a termenilor de forma Termen

```
care satisfac conditia Scop; in cazul in care nu exista astfel de termeni, findall intoarce lista
vida: Lista=[]; mai exista o diferenta intre bagof si findall, in cazul in care conditia Scop
include variabile care nu apar in Termen - vom vedea.
*/

:- op(300,xfx,are).

anca are pere.
anca are Fructe :- nicu are Fructe.

/* Interogati:
?- anca are Ce.
?- maria are Ce.
?- Cine are Ce.
?- setof(Cine are Ce, are(Cine,Ce), L), write(L), length(L,CateSolutii).
?- setof(Cine are Ce, Cine are Ce, L), write(L), length(L,CateSolutii).
?- bagof(Cine are Ce, Cine are Ce, L), write(L), length(L,CateSolutii).
?- findall(Cine are Ce, Cine are Ce, L), write(L), length(L,CateSolutii).
?- Cine are visine.
?- setof(Cine, Cine are visine, L), write(L), length(L,CateSolutii).
?- bagof(Cine, Cine are visine, L), write(L), length(L,CateSolutii).
?- findall(Cine, Cine are visine, L), write(L), length(L,CateSolutii).
*/

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/*
?- bradut(5).
    *
   * *
  * * *
```

```
 * * * *
* * * * *
*/

bradut(N) :- integer(N), N>=0, brad(N).

brad(N) :- linii(N,0).

linii(0,_).
linii(N,K) :- N>0, PN is N-1, SK is K+1, linii(PN,SK), nl, tab(K), stelute(N).

stelute(0).
stelute(N) :- N>0, write(*), tab(1), PN is N-1, stelute(PN).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/*
[elem1,elem2,...,elemn|T]=[elem1|[elem2,...,elemn|T]]=[|](elem1,[elem2,...,elemn|T])
Predicat predefinit: =..
Termen =.. Lista
Termen =.. [OperatorDominantTermen|ListaArgumenteTermen]
In documentatia Prolog-ului, in descrierea predicatelor predefinite:
   argumentele precedate de + trebuie furnizate Prolog-ului in interogari;
   argumentele precedate de - pot fi calculate de Prolog in interogari;
   argumentele precedate de ? pot avea oricare dintre rolurile de mai sus.
*/

% Prolog-ul permite supraincarcarea operatorilor.

f.
f(X) :- X>10.
```

```
f(X,Y) :- f(X), not(f(Y)).
f(_,_,_).

/* Interogati:
?- f([],f(0),h(f(1),g(2),h(3)))=..L.
?- f([],f(0),h(f(1),g(2),h(3)))=..[Op|LA], length(LA,NrArg).
?- Termenul=..[f,2,g(3),[],[a,b,1,c]].
?- [1,2,3]=..[Op|LA], length(LA,Aritate).
?- [2,3]=..[Op|LA], length(LA,Aritate).
?- [3]=..[Op|LA], length(LA,Aritate).
?- []=..[Op|LA], length(LA,Aritate).
?- c=..[Op|LA], length(LA,Aritate).
?- f(X,1,a,[2,3],g(V))=..[Op|LA], length(LA,Aritate).
?- length([a,b,c],NrElem).
?- length(CeFelDeLista,3).
*/

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Predicat echivalent cu predicatul predefinit length:

lungime([],0).
lungime([_|T],N) :- lungime(T,K), N is K+1.

/* Cu varianta:
lungime([_|T],N) :- lungime(T,K), N=K+1.
intrucat [a,b,c]=[a|[b|[c|[]]]], am obtine la interogarea:
?- lungime([a,b,c],Care).
      +
     / \
    +   1
```

```
   / \
  +   1
 / \
0   1
*/

% Predicat echivalent cu predicatul predefinit append pentru concatenare de liste:

concat([],L,L).
concat([H|T],L,[H|M]) :- concat(T,L,M).

/* Interogati:
?- concat([a,b],[1,2,3],L).
?- concat(L,[1,2,3],[a,b,1,2,3]).
?- concat([a,b],L,[a,b,1,2,3]).
?- concat(L,CuCeL,[a,b,1,2,3]).
*/

% Predicat echivalent cu predicatul predefinit reverse pentru inversare de liste:

inversa([],[]).
inversa([H|T],L) :- inversa(T,M), concat(M,[H],L).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/* Predicatul cut: !  are rolul de a taia backtracking-ul efectuat de interpretorul de Prolog (mai
precis din algoritmul sau backward chaining pe care acest interpretor il are incorporat, ai carui
pasi constau din aplicarea cate unei reguli de rezolutie in logica clasica a predicatelor, pentru un
caz particular dat de forma clauzelor din Prolog; deductia folosind regula rezolutiei in logica
propozitionala clasica este echivalenta cu deductia prin sistemul Hilbert (A1,A2,A3,MP) pe care il
vom studia la inceputul acelui capitol al cursului; pentru logica clasica a predicatelor, in general
```

```
regula rezolutiei nu este echivalenta cu deductia data de sistemul Hilbert al acestei logici (= cel
al logicii propozitionale clasice extins cu cateva axiome si o regula de deductie), dar este
echivalenta cu acest sistem Hilbert in cazul particular in care este aplicata de Prolog),
determinand Prolog-ul sa nu mai caute alte solutii ale scopului curent din momentul in care
intalneste predicatul cut.
Ca exemple, dati urmatoarele interogari:
?- true; true.
?- prop; altaprop.
?- true, !; true.
?- prop, !; altaprop.
?- not(prop), !; altaprop.
?- prop, write(***), !; write('am ajuns la membrul drept'), altaprop.
?- not(prop), write(***), !; write('am ajuns la membrul drept'), altaprop.
?- prop, write('ajung la cut'), !; write('am ajuns la membrul drept'), altaprop.
?- not(prop), write('ajung la cut'), !; write('am ajuns la membrul drept'), altaprop.
*/

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

apartine(_,[]) :- fail.  % echivalent: not(apartine(_,[])).
apartine(H,[H|_]) :- !.
apartine(H,[_|T]) :- apartine(H,T).

% Predicat echivalent cu predicatul predefinit member:

membru(_,[]) :- fail.
membru(H,[H|_]). % echivalent cu: membru(H,[X|_]) :- H=X.
membru(H,[_|T]) :- membru(H,T).

% Aparitiile literal identice ale unui element intr-o lista:
```

```
membrulitid(_,[]) :- fail.
membrulitid(H,[X|_]) :- H==X.
membrulitid(H,[_|T]) :- membrulitid(H,T).

/* Interogati:
?- apartine(Cine,[a,b,c]).
?- membru(Cine,[a,b,c]).
?- membru(Cine,[a,b,V,c,W]).
?- membru(Cine,[a,Cine,b,Cine,V,c,W,Cine]).
?- membrulitid(Cine,[a,b,c]).
?- membrulitid(Cine,[a,b,V,c,W]).
?- membrulitid(Cine,[a,Cine,b,Cine,V,c,W,Cine]).
*/

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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

elimdupl([],[]).
elimdupl([H|T],[H|L]) :- sterge(H,T,M), elimdupl(M,L).

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

detdupl([],[]).
detdupl([H|T],[H|L]) :- member(H,T), !, sterge(H,T,M), detdupl(M,L).
detdupl([_|T],L) :- detdupl(T,L).
```

```
/* Interogati:
?- setof(X, member(X,[1,a,0,b,a,c,0,1,a,1,1]), L).
?- elimdup([1,a,0,b,a,c,0,1,a,1,1], L).
?- elimdupl([1,a,0,b,a,c,0,1,a,1,1], L).
?- detdupl([1,a,0,b,a,c,0,1,a,1,1], L).
?- findall(Cine are Ce, Cine are Ce, L), write('Lista solutiilor: '), write(L),
length(L,CateSolutii), detdupl(L,SolutiiDuplicate), nl, write('Lista solutiilor duplicate: '),
write(SolutiiDuplicate), length(SolutiiDuplicate,CateSolutiiDuplicate).
?- setof((X,Y), (member(X,[a,b]), member(Y,[1,2,3])), L).
?- setof((X,Y), (member(X,[a,b]), member(Y,[1,2,2])), L).
?- bagof((X,Y), (member(X,[a,b]), member(Y,[1,2,2])), L).
?- setof((X,Y), (member(X,[]), member(Y,[1,2,2])), L).
?- bagof((X,Y), (member(X,[]), member(Y,[1,2,2])), L).
?- findall((X,Y), (member(X,[]), member(Y,[1,2,2])), L).
*/

% Produsul cartezian de multimi, deci fara duplicate:

prodcartmult(M,N,P) :- setof((X,Y), (member(X,M), member(Y,N)), P), !.
prodcartmult(_,_,[]).

/* Interogati:
?- prodcartmult([a,b],[1,2,3],P).
?- prodcartmult([a,b],[1,2,2],P).
?- prodcartmult([],[1,2,2],P).
*/

/* Sa demonstram distributivitatea disjunctiei fata de conjunctie, adica faptul ca,
pentru orice valori de adevar ale enunturilor p, q, r, avem:
[p sau (q si r)] <=> [(p sau q) si (p sau r)],
i.e. enunturile compuse [p sau (q si r)] si [(p sau q) si (p sau r)] au aceeasi valoare de adevar.
```

```
Sa observam ca: [p=>q] <=> [non p sau q], unde conectorul unar non e considerat mai prioritar decat
orice conector binar, in particular decat sau, asadar [non p sau q] inseamna [(non p) sau q] si nu
[non (p sau q)]. */

implica(P,Q) :- not(P); Q.
echiv(P,Q) :- implica(P,Q), implica(Q,P).

ms(P,Q,R) :- P ; Q, R.
md(P,Q,R) :- (P ; Q), (P ; R).

dedem(P,Q,R) :- echiv(ms(P,Q,R),md(P,Q,R)).

dedem :- not((member(P,[false,true]), member(Q,[false,true]), member(R,[false,true]),
        write((P,Q,R)), nl, not(echiv(ms(P,Q,R),md(P,Q,R))))).

sunttoate(L,SperCaSuntToateOpt) :- setof((P,Q,R), (member(P,[false,true]), member(Q,[false,true]),
member(R,[false,true]), echiv(ms(P,Q,R),md(P,Q,R))), L), write(L), length(L,SperCaSuntToateOpt).

testsunttoate :- setof((P,Q,R), (member(P,[false,true]), member(Q,[false,true]),
member(R,[false,true]), echiv(ms(P,Q,R),md(P,Q,R))), L), length(L,SperCaSuntToateOpt).

sunttoate :- setof((P,Q,R), (member(P,[false,true]), member(Q,[false,true]), member(R,[false,true]),
echiv(ms(P,Q,R),md(P,Q,R))), L), write(L), nl, length(L,SperCaSuntToateOpt),
write('SperCaSuntToateOpt = '), write(SperCaSuntToateOpt).

/* Interogati:
?- dedem.
?- sunttoate(L,SperCaSuntToateOpt).
?- sunttoate.
*/
```