```
/* LABORATORIUM 5 */


/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */
/* Zadanie 1 */

% dane

rodzina(
    osoba(jan, kowalski,data(5,kwiecien,1946),pracuje(tpsa,3000)),
    osoba(anna,kowalski,data(8,luty,1949),    pracuje(szkola,1500)),
    [
     osoba(maria,kowalski,data(20,maj,1973),     pracuje(argo_turist,4000)),
     osoba(pawel,kowalski,data(15,listopad,1979),zasilek)]).

rodzina(
    osoba(krzysztof, malinowski, data(24,lipiec,1950), bezrobocie),
    osoba(klara, malinowski, data(9,styczen,1951), pracuje(kghm,8000)),
    [
     osoba(monika, malinowski, data(19,wrzesien,1980), bezrobocie)]
    ).

% zaleznosci

maz(X) :-
    rodzina(X,_,_).

zona(X) :-
    rodzina(_,X,_).

dziecko(X) :-
    rodzina(_,_,Dzieci),
    nalezy(X,Dzieci).

istnieje(Osoba) :-
    maz(Osoba)
    ;
    zona(Osoba)
    ;
    dziecko(Osoba).

data_urodzenia(osoba(_,_,Data,_),Data).

pensja(osoba(_,_,_,pracuje(_,P)),P).
pensja(osoba(_,_,_,zasilek),500).
pensja(osoba(_,_,_,bezrobocie),0).

zarobki([],0).
zarobki([Osoba|Lista],Suma) :-
    pensja(Osoba,S),
    zarobki(Lista,Reszta),
    Suma is S + Reszta.

% narzedzia
nalezy(X,[X|_]).
nalezy(X,[_|Yogon]) :-
    nalezy(X,Yogon).




/* Pobrac program r-fam.pl
Przegladnac calosc,
Zadac pytania : */


% Jakie sa osoby w bazie ?
istnieje(X).

% Jakie sa dzieci w bazie ?
dziecko(X).

% Pokazac pensje wszystkich osob.
istnieje(X),pensja(X,P).

% Jakie dzieci urodzily sie w 1979r ?
dziecko(X),data_urodzenia(X,data(_,_,Y)),Y == 1979.

% Znalezc wszystkie zony, ktore pracuja.
zona(osoba(I,N,_,pracuje(_,_))).

% Znalezc osoby urodzone przed 1950 r, ktorych pensja jest rowna 3000
istnieje(osoba(I,N,data(_,_,R),pracuje(_,P))),R<1950,P==3000.

% Korzystajac z wiedzy zdobytej na poprzednim laboratorium, prosze policzyc
% zarobki wszystkich osob:
bagof(X,istnieje(X),L),zarobki(L,Z).

% Dopisac kolejne dwie rodziny do reprfam.pl

/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */
```

```
/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */
/* Zadanie 2 */

% Copyright by Ivan Bratko
% Figure 4.5  A flight route planner and an example flight timetable.
%


% A FLIGHT ROUTE PLANNER

:-  op( 50, xfy, :).

% route( Place1, Place2, Day, Route):
%    Route is a sequence of flights on Day, starting at Place1, ending at Place2

route( P1, P2, Day, [ P1 / P2 / Fnum / Deptime ] )  :-    % Direct flight
  flight( P1, P2, Day, Fnum, Deptime, _).

route( P1, P2, Day, [ (P1 / P3 / Fnum1 / Dep1) | RestRoute] )  :-    % Indirect connection
  route( P3, P2, Day, RestRoute),
  flight( P1, P3, Day, Fnum1, Dep1, Arr1),
  deptime( RestRoute, Dep2),                                        % Departure time of Route
  transfer( Arr1, Dep2).                                        % Enough time for transfer

flight( Place1, Place2, Day, Fnum, Deptime, Arrtime)  :-
   timetable( Place1, Place2, Flightlist),
   member( Deptime / Arrtime / Fnum / Daylist , Flightlist),
   flyday( Day, Daylist).

flyday( Day, Daylist)  :-
   member( Day, Daylist).

flyday( Day, alldays)  :-
   member( Day, [mo,tu,we,th,fr,sa,su] ).

deptime( [ _ / _ / _ / Dep | _], Dep).

transfer( Hours1:Mins1, Hours2:Mins2)  :-
   60 * (Hours2 - Hours1) + Mins2 - Mins1 >= 40.

member( X, [X | _] ).

member( X, [_ | L] )  :-
   member( X, L).


% A FLIGHT DATABASE

timetable( edinburgh, london,
           [  9:40 / 10:50 / ba4733 / alldays,
             13:40 / 14:50 / ba4773 / alldays,
             19:40 / 20:50 / ba4833 / [mo,tu,we,th,fr,su] ] ).

timetable( london, edinburgh,
           [  9:40 / 10:50 / ba4732 / alldays,
             11:40 / 12:50 / ba4752 / alldays,
             18:40 / 19:50 / ba4822 / [mo,tu,we,th,fr] ] ).

timetable( london, ljubljana,
           [ 13:20 / 16:20 / jp212 / [mo,tu,we,fr,su],
             16:30 / 19:30 / ba473 / [mo,we,th,sa] ] ).

timetable( london, zurich,
           [  9:10 / 11:45 / ba614 / alldays,
             14:45 / 17:20 / sr805 / alldays ] ).

timetable( london, milan,
           [  8:30 / 11:20 / ba510 / alldays,
             11:00 / 13:50 / az459 / alldays ] ).

timetable( ljubljana, zurich,
           [ 11:30 / 12:40 / jp322 / [tu,th] ] ).

timetable( ljubljana, london,
           [ 11:10 / 12:20 / jp211 / [mo,tu,we,fr,su],
             20:30 / 21:30 / ba472 / [mo,we,th,sa] ] ).

timetable( milan, london,
           [  9:10 / 10:00 / az458 / alldays,
             12:20 / 13:10 / ba511 / alldays ] ).

timetable( milan, zurich,
           [  9:25 / 10:15 / sr621 / alldays,
             12:45 / 13:35 / sr623 / alldays ] ).

timetable( zurich, ljubljana,
           [ 13:30 / 14:40 / jp323 / [tu,th] ] ).
```

```
timetable( zurich, london,
           [ 9:00 / 9:40 / ba613 / [mo,tu,we,th,fr,sa],
            16:10 / 16:55 / sr806 / [mo,tu,we,th,fr,su] ] ).

timetable( zurich, milan,
           [ 7:55 / 8:45 / sr620 / alldays ] ).


query3(City1,City2,City3,FN1,FN2,FN3,FN4)  :-
  permutation( [milan,ljubljana,zurich],[City1,City2,City3]),
  flight( london, City1, tu, FN1, _, _),
  flight( City1, City2, we, FN2, _, _),
  flight( City2, City3, th, FN3, _, _),
  flight( City3, london, fr, FN4, _, _).

conc([], L, L).

conc([X|L1],L2, [X|L3]) :-
  conc(L1,L2,L3).

permutation( [], []).

permutation( L, [X | P])  :-
  del( X, L, L1),
  permutation( L1, P).

del( X, [X|L], L).

del( X, [Y|L], [Y|L1])  :-
  del( X, L, L1).



% Pobrac program flight_planner-1.pl

% Przetestowac, np.:

flight(ljubljana,london,Dzien,_,Wylot:_,_),Wylot >= 18.
% W ktore dni tygodnia mamy bezposredni lot z Ljubljany do Londynu, po 18 ?

route(ljubljana, edinburgh,th,R).
% Jak mozna dostac sie z Ljubljany do Edynburga w czwartek ?


/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */



/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */
/* Zadanie 3 */

% Predykaty display, write_canonical wypisuja Prologowa reprezentacje termów
% Dlugie listy wygodnie jest wypisywac przy uzyciu print

A=1, display(A).

1
A = 1.


A = [1], display(A).

[1]
A = [1].


A = [1], write_canonical(A).

[1]
A = [1].


A = [1,a,[ala,ma,[kota]]], display(A).

[1,a,[ala,ma,[kota]]]
A = [1, a, [ala, ma, [kota]]].


X = [1,2,3,45,6,7,8,9,32,4,6,ff,7,d],print(X).

[1,2,3,45,6,7,8,9,32,4,6,ff,7,d]
X = [1, 2, 3, 45, 6, 7, 8, 9, 32|...].

/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */


/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */
/* Zadanie 4 */

% Zastosowanie cut:

a(1).
```

```
c(1).
c(2).

a(X) :- c(X), !.

?- a(X), write(X), fail.
1
false.

% Dla porownania bez cat:

c(1).
c(2).

a(X) :- c(X).


?- a(X), write(X), fail.
12
false.


% Cut odcina mozliwosc cofania sie programu do predykatow
% ktore sa przed nim

[trace]  ?- a(1).
   Call: (7) a(1) ? creep
   Call: (8) c(1) ? creep
   Exit: (8) c(1) ? creep
   Call: (8) d(1) ? creep
   Fail: (8) d(1) ? creep
   Fail: (7) a(1) ? creep
false.

[trace]  ?- a(2).
   Call: (7) a(2) ? creep
   Call: (8) c(2) ? creep
   Exit: (8) c(2) ? creep
   Call: (8) d(2) ? creep
   Exit: (8) d(2) ? creep
   Exit: (7) a(2) ? creep
true.

[trace]  ?- a(X).
   Call: (7) a(_G1266) ? creep
   Call: (8) c(_G1266) ? creep
   Exit: (8) c(1) ? creep
   Call: (8) d(1) ? creep
   Fail: (8) d(1) ? creep
   Fail: (7) a(_G1266) ? creep
false.



% Pomimo wolania fail i ponowenego wejscia w a, nie mozna zmienic wartosci X
c(1).
c(2).
c(3).
d(1).
d(2).
d(3).

a(X,Y) :- c(X), !, d(Y).

?- a(X,Y), write('X = '), write(X), write(', Y = '), writeln(Y),  fail.
X = 1, Y = 1
X = 1, Y = 2
X = 1, Y = 3
false.


% Testowanie predykatu nalezy

nalezy(X, [X|Reszta]) :- write(X).

nalezy(Element, [_|Ogon]) :-
    nalezy(Element, Ogon).

?- nalezy(a,[a,b,c]),fail.
a
false.

?- nalezy(a,[a,b,a,c]),fail.
aaaaa
false.

% uzywajac cut:

nalezy(X, [X|Reszta]) :- !, write(X).

nalezy(Element, [_|Ogon]) :-
    nalezy(Element, Ogon).
```

```
    nalezy(Element, Ogon).

?- nalezy(a,[a,b,c]),fail.
a
false.

?- nalezy(a,[a,b,a,c]),fail.
aaa
false.

% predykat max z uzyciem i bez uzycia cut
% W pierwszym przypadku zawsze wykonają się dwa predykaty
% w drugium wykona się tylko pierwsze jeśli X >= Y
max1(X,Y,X) :- X >= Y.
max1(X,Y,Y) :- X < Y.

max2(X,Y,X) :- X >= Y, !.
max2(_,Y,Y).

[trace]  ?- max1(2,3,X).
   Call: (7) max1(2, 3, _G2123) ? creep
   Call: (8) 2>=3 ? creep
   Fail: (8) 2>=3 ? creep
   Redo: (7) max1(2, 3, _G2123) ? creep
   Call: (8) 2<3 ? creep
   Exit: (8) 2<3 ? creep
   Exit: (7) max1(2, 3, 3) ? creep
X = 3.

[trace]  ?- max2(2,3,X).
   Call: (7) max2(2, 3, _G2123) ? creep
   Call: (8) 2>=3 ? creep
   Fail: (8) 2>=3 ? creep
   Redo: (7) max2(2, 3, _G2123) ? creep
   Exit: (7) max2(2, 3, 3) ? creep
X = 3.

/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */


/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */
% Zastosowanie cut:

c(1).
c(2).

a(X) :- c(X), !.

?- a(X), write(X), fail.
1
false.

% Dla porownania bez cat:

c(1).
c(2).

a(X) :- c(X).


?- a(X), write(X), fail.
12
false.


% Cut odcina mozliwosc cofania sie programu do predykatow
% ktore sa przed nim

[trace]  ?- a(1).
   Call: (7) a(1) ? creep
   Call: (8) c(1) ? creep
   Exit: (8) c(1) ? creep
   Call: (8) d(1) ? creep
   Fail: (8) d(1) ? creep
   Fail: (7) a(1) ? creep
false.

[trace]  ?- a(2).
   Call: (7) a(2) ? creep
   Call: (8) c(2) ? creep
   Exit: (8) c(2) ? creep
   Call: (8) d(2) ? creep
   Exit: (8) d(2) ? creep
   Exit: (7) a(2) ? creep
true.

[trace]  ?- a(X).
   Call: (7) a(_G1266) ? creep
   Call: (8) c(_G1266) ? creep
   Exit: (8) c(1) ? creep
   Call: (8) d(1) ? creep
   Fail: (8) d(1) ? creep
```

```
   Fail: (7) a(_G1266) ? creep
false.



% Pomimo wolania fail i ponowenego wejscia w a, nie mozna zmienic wartosci X
c(1).
c(2).
c(3).
d(1).
d(2).
d(3).

a(X,Y) :- c(X), !, d(Y).

?- a(X,Y), write('X = '), write(X), write(', Y = '), writeln(Y),  fail.
X = 1, Y = 1
X = 1, Y = 2
X = 1, Y = 3
false.


% Testowanie predykatu nalezy

nalezy(X, [X|Reszta]) :- write(X).

nalezy(Element, [_|Ogon]) :-
    nalezy(Element, Ogon).

?- nalezy(a,[a,b,c]),fail.
a
false.

?- nalezy(a,[a,b,a,c]),fail.
aaaaa
false.

% uzywajac cut:

nalezy(X, [X|Reszta]) :- !, write(X).

nalezy(Element, [_|Ogon]) :-
    nalezy(Element, Ogon).

?- nalezy(a,[a,b,c]),fail.
a
false.

?- nalezy(a,[a,b,a,c]),fail.
aaa
false.

% predykat max z uzyciem i bez uzycia cut
% W pierwszym przypadku zawsze wykonają się dwa predykaty
% w drugium wykona się tylko pierwsze jeśli X >= Y
max1(X,Y,X) :- X >= Y.
max1(X,Y,Y) :- X < Y.

max2(X,Y,X) :- X >= Y, !.
max2(_,Y,Y).

[trace]  ?- max1(2,3,X).
   Call: (7) max1(2, 3, _G2123) ? creep
   Call: (8) 2>=3 ? creep
   Fail: (8) 2>=3 ? creep
   Redo: (7) max1(2, 3, _G2123) ? creep
   Call: (8) 2<3 ? creep
   Exit: (8) 2<3 ? creep
   Exit: (7) max1(2, 3, 3) ? creep
X = 3.

[trace]  ?- max2(2,3,X).
   Call: (7) max2(2, 3, _G2123) ? creep
   Call: (8) 2>=3 ? creep
   Fail: (8) 2>=3 ? creep
   Redo: (7) max2(2, 3, _G2123) ? creep
   Exit: (7) max2(2, 3, 3) ? creep
X = 3.
/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */


/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */
% Zagadnia einsteina
% Sprawdzenie, czy element Lewo znajduje się na lewo od elementu Prawo w liście.
na_lewo(Lewy,Prawy,[Lewy,Prawy|_]).
na_lewo(Lewy,Prawy,[_|Ogon]) :- na_lewo(Lewy,Prawy,Ogon).

% Sprawdzenie, czy elementy X i Y znajdują się obok siebie w liście Domy.
obok(X,Y,Domy) :- na_lewo(X,Y,Domy).
obok(X,Y,Domy) :- na_lewo(Y,X,Domy).
```

```
% Rozwiązanie zagadki.
rozwiazanie(Szukany) :-
    % 5 ludzi różnych narodowości zamieszkuje 5 domów w 5 różnych kolorach.
    % Wszyscy palą papierosy 5 różnych marek i piją 5 różnych napojów.
    % Hodują zwierzęta 5 różnych gatunków.
    % Lista reprezentująca pojedynczy dom:
    % [numer,kolor,mieszkaniec,papierosy,napoj,zwierzeta].
    Domy = [[1,_,_,_,_,_],[2,_,_,_,_,_],[3,_,_,_,_,_],[4,_,_,_,_,_],[5,_,_,_,_,_]],
    % Norweg zamieszkuje pierwszy dom.
    member([1,_,norweg,_,_,_],Domy),
    % Anglik mieszka w czerwonym domu.
    member([_,czerwony,anglik,_,_,_],Domy),
    % Zielony dom znajduje się bezpośrednio po lewej stronie domu białego.
    na_lewo([_,zielony,_,_,_,_],[_,bialy,_,_,_,_],Domy),
    % Duńczyk pije herbatkę.
    member([_,_,dunczyk,_,herbata,_],Domy),
    % Palacz Rothmansów mieszka obok hodowcy kotów.
    obok([_,_,_,rothmans,_,_],[_,_,_,_,_,koty],Domy),
    % Mieszkaniec żółtego domu pali Dunhille.
    member([_,zolty,_,dunhill,_,_],Domy),
    % Niemiec pali Marlboro.
    member([_,_,niemiec,marlboro,_,_],Domy),
    % Mieszkaniec środkowego domu pija mleko.
    member([3,_,_,_,mleko,_],Domy),
    % Palacz Rothmansów ma sąsiada, który pija wodę.
    obok([_,_,_,rothmans,_,_],[_,_,_,_,woda,_],Domy),
    % Palacz Pall Malli hoduje ptaki.
    member([_,_,_,pall_mall,_,ptaki],Domy),
    % Szwed hoduje psy.
    member([_,_,szwed,_,_,psy],Domy),
    % Norweg mieszka obok niebieskiego domu.
    obok([_,_,norweg,_,_,_],[_,niebieski,_,_,_,_],Domy),
    % Hodowca koni mieszka obok żółtego domu.
    obok([_,_,_,_,_,konie],[_,zolty,_,_,_,_],Domy),
    % Palacz Philip Morris pija piwo.
    member([_,_,_,phillip_morris,piwo,_],Domy),
    % W zielonym domu pija się kawę.
    member([_,zielony,_,_,kawa,_],Domy),
    % Szukamy mieszkańca domu, w którym hoduje się rybki.
    member([_,_,Szukany,_,_,rybki],Domy).

    1 ?- rozwiazanie(Hodowca_rybek).
    Hodowca_rybek = niemiec ;
    fail.

    1 ?- rozwiazanie(S).
    S = [norweg, dunhill] ;
    S = [dunczyk, rothmans] ;
    S = [anglik, pall_mall] ;
    S = [niemiec, marlboro] ;
    S = [szwed, phillip_morris] ;
    fail.

/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */


/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */
/* Sortowania */

% bratko, 9.1

gt(X,Y) :- X>Y.
gt(X,Y) :- X@>Y.

bubblesort(List, Sorted) :-
    swap(List, List1), !,
    bubblesort(List1, Sorted).
    % A useful swap in List ?

bubblesort(Sorted, Sorted).
    % Otherwise list is already sorted

swap([X,Y|Rest], [Y,X|Rest]) :- gt(X, Y).
    % swap first two elements

swap([Z|Rest], [Z|Rest1]) :- swap(Rest, Rest1).
    % swap elements in tail



% bratko 9.1

gt(X,Y) :- X>Y.
gt(X,Y) :- X@>Y.

insertsort([], []).
insertsort([Head|Tail], L2):-
        insertsort(Tail, L1),
        insert(Head, L1, L2).

insert(Element, [Head|Tail], [Head|Rest]):-
```

```
        gt(Element,Head),!,
        insert(Element, Tail, Rest).
insert(Element, List, [Element|List]).




% From the book
% PROLOG PROGRAMMING IN DEPTH
% by Michael A. Covington, Donald Nute, and Andre Vellino
% (Prentice Hall, 1997).
% Copyright 1997 Prentice-Hall, Inc.
% For educational use only

% File MSORT.PL
% Mergesort


% mergesort(+List1,-List2)
%  Sorts List1 giving List2 using mergesort.

mergesort([First,Second|Rest],Result) :-   % list has at least 2 elements
  !,
  partition([First,Second|Rest],L1,L2),
  mergesort(L1,SL1),
  mergesort(L2,SL2),
  mergelist(SL1,SL2,Result).

mergesort(List,List).                      % list has 0 or 1 element



% mergelist(+List1,+List2,-Result)
%  Combines two sorted lists into a sorted list.

mergelist([First1|Rest1],[First2|Rest2],[First1|Rest]) :-
  First1 @< First2,
  !,
  mergelist(Rest1,[First2|Rest2],Rest).

mergelist([First1|Rest1],[First2|Rest2],[First2|Rest]) :-
  \+ First1 @< First2,
  !,
  mergelist([First1|Rest1],Rest2,Rest).

mergelist(X,[],X).

mergelist([],X,X).



% partition(+List,-List1,-List2)
%  splits List in two the simplest way,
%  by putting alternate members in different lists

partition([First,Second|Rest],[First|F],[Second|S]) :-  % 2 or more elements
   !,
   partition(Rest,F,S).

partition(List,List,[]).                                % 0 or 1 element



% Demonstration predicate

testsort :- mergesort([7,0,6,5,4,9,4,6,3,3],What), write(What).




% Figure 9.2  Quicksort.


% quicksort( List, SortedList): sort List by the quicksort algorithm

gt(X,Y) :- X>Y.
gt(X,Y) :- X@>Y.

conc([],L,L).
conc([X|L1],L2,[X|L3]) :-
    conc(L1,L2,L3).

quicksort( [], []).

quicksort( [X|Tail], Sorted)  :-
   split( X, Tail, Small, Big),
   quicksort( Small, SortedSmall),
   quicksort( Big, SortedBig),
   conc( SortedSmall, [X|SortedBig], Sorted).

split(_,[],[],[]).
```

```
split( _, [], [], []).

split( X, [Y|Tail], [Y|Small], Big)  :-
   gt( X, Y), !,
   split( X, Tail, Small, Big).

split( X, [Y|Tail], Small, [Y|Big])  :-
   split( X, Tail, Small, Big).
/* -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  -- */
```