prolog

lab 1

```
kobieta(maria).
kobieta(ewa).
kobieta(agata).
kobieta(anna).
kobieta(joanna).
kobieta(agnieszka).
kobieta(beata).
kobieta(iwona).

mężczyzna(piotr).
mężczyzna(adam).
mężczyzna(marek).
mężczyzna(robert).
mężczyzna(jan).
mężczyzna(krzysztof).
mężczyzna(radek).
mężczyzna(darek).
mężczyzna(tomek).
mężczyzna(jacek).

rodzic(maria,marek).
rodzic(piotr,marek).
rodzic(adam,joanna).
rodzic(ewa,joanna).
rodzic(maria,agata).
rodzic(piotr,agata).
rodzic(maria,jan).
rodzic(piotr,jan).
rodzic(adam,anna).
rodzic(ewa,anna).
rodzic(adam,krzysztof).
rodzic(ewa,krzysztof).
rodzic(robert,radek).
rodzic(agata,radek).
rodzic(robert,beata).
rodzic(agata,beata).
rodzic(jan,darek).
rodzic(anna,darek).
rodzic(jan,tomek).
rodzic(anna,tomek).
rodzic(krzysztof,jacek).
rodzic(agnieszka,jacek).
rodzic(krzysztof,iwona).
rodzic(agnieszka,iwona).
```

małżeństwo(maria,piotr).
małżeństwo(adam,ewa).
małżeństwo(robert,agata).
małżeństwo(jan,anna).
małżeństwo(krzysztof,agnieszka).

matka(X,Y) :- kobieta(X), rodzic(X,Y).
ojciec(X,Y) :- mężczyzna(X), rodzic(X,Y).
brat(X,Y) :- mężczyzna(X), rodzic(Z,X), rodzic(Z,Y), X\=Y.
siostra(X,Y) :- kobieta(X), rodzic(Z,X), rodzic(Z,Y), X\=Y.
babcia(X,Y) :- kobieta(X), rodzic(X,Z), rodzic(Z,Y).
dziadek(X,Y) :- mężczyzna(X), rodzic(X,Z), rodzic(Z,Y).
wuj(X,Y) :- matka(Z,Y), brat(X,Z).
stryj(X,Y) :- ojciec(Z,Y), brat(X,Z).
kuzyn(X,Y) :- mężczyzna(X), rodzic(Z,Y), rodzic(Q,X), rodzic(W,Z), rodzic(W,Q), X\=Y, Z\=Q.
kuznynka(X,Y) :- kobieta(X), rodzic(Z,Y), rodzic(Q,X), rodzic(W,Z), rodzic(W,Q), X\=Y, Z\=Q.
teściowa(X, Y) :- małżeństwo(Y, Z), kobieta(Z), matka(X, Z).
teściowa(X, Y) :- małżeństwo(Z, Y), kobieta(Z), matka(X, Z).
żona(X, Y) :- małżeństwo(X, Y), kobieta(X).
żona(X, Y) :- małżeństwo(Y, X), kobieta(X).
mąż(X, Y) :- małżeństwo(X, Y), mężczyzna(X).
mąż(X, Y) :- małżeństwo(Y, X), mężczyzna(X).
szwagier(X, Y) :- żona(Z, Y), brat(X, Z).
szwagier(X, Y) :- siostra(Z, Y), mąż(X, Z).

lab 2

```
pionowy(odcinek(punkt(X, _), punkt(X,_))).
poziomy(odcinek(punkt(_, Y), punkt(_,Y))).

regularny(prostokąt(punkt(X1,Y1),punkt(X2,Y1), punkt(X2, Y2), punkt(X1,Y2))).




%F, W, G, C
%
%



safe(state(X,_,X,_)).
safe(state(X,X,_,X)).




move(state(east,W,G,C), wf, state(west,W,G,C)):-
    safe(state(west,W,G,C)).

move(state(east,east,G,C), wfw, state(west,west,G,C)):-
    safe(state(west,west,G,C)).


move(state(east,W,east,C), wfg, state(west,W,west,C)):-
    safe(state(west,W,west,C)).

move(state(east,W,G,east), wfc, state(west,W,G,west)):-
    safe(state(west,W,G,west)).

move(state(west,W,G,west), efc, state(east,W,G,east)):-
    safe(state(east,W,G,east)).


move(state(west,west,G,C), efw, state(east,east,G,C)):-
     safe(state(east,east,G,C)).

move(state(west,W,west,C), efg, state(east,W,east,C)):-
    safe(state(east,W,east,C)).


move(state(west,W,G,C), ef, state(east,W,G,C)):-
     safe(state(east,W,G,C)).
```

```
can_get(state(east,east,east,east)).
can_get(StateBefore) :- move(StateBefore, Dir ,StateAfter), can_get(StateAfter).
```

lab 3

```
next(świnoujście,kołobrzeg).
next(świnoujście, szczecin).
next(kołobrzeg,ustka).
next(ustka, gdańsk).
next(gdańsk,toruń).
next(gdańsk,olsztyn).
next(olsztyn,toruń).
next(olsztyn,białystok).
next(szczecin,bydgoszcz).
next(szczecin, gorzów-wlkp).
next(bydgoszcz,toruń).
next(bydgoszcz,poznań).
next(poznań,toruń).
next(gorzów-wlkp,poznań).
next(gorzów-wlkp,zielona-góra).
next(zielona-góra,poznań).
next(zielona-góra,wrocław).
next(toruń,łódź).
next(toruń,warszawa).
next(białystok, warszawa).
next(białystok,lublin).
next(poznań,łódź).
next(warszawa,radom).
next(warszawa,łódź).
next(radom,lublin).
next(radom,kielce).
next(wrocław,łódź).
next(łódź,radom).
next(łódź,częstochowa).
next(częstochowa,kielce).
next(częstochowa,katowice).
next(katowice,kraków).
next(kielce,kraków).
next(kielce,rzeszów).
next(lublin,kielce).
next(lublin,rzeszów).
next(kraków,rzeszów).
% Pytanie: Czy istnieje połączenie miedzy Świnoujściem a Rzeszowem?
połączenie(From, To) :- next(From, To).
połączenie(From, To) :- next(From, Pit), połączenie(Pit, To).



zjazd(From, To, By) :- next(From, To), From = By.
zjazd(From, To, By) :- next(From, Pit), zjazd(Pit, To, By).
```

```
trasa(From, To, obok(From, To)) :- next(From, To).
trasa(From, To, obok(From, W)) :- next(From, Pit), trasa(Pit, To, W).


trasa2(From, To, obok(From, To)) :- next(From, To).
trasa2(From, To, obok(W, To)) :- next(Pit, To), trasa2(From, Pit, W).

trasa3(From, To, [From|[To|[]]]) :- next(From, To).
trasa3(From, To, [From|W]) :- next(From, Pit), trasa3(Pit, To, W).
```

lab 4

```
ostatni(Last, [Last]).
ostatni(Member, [Head|Tail]) :- ostatni(Member, Tail).

przedostatni(Prev2last, [Prev2last, _]).
przedostatni(Member, [Head1, Head2 | Tail]) :- przedostatni(Member, [Head2 | Tail]).

bezostatniego([_], []).
bezostatniego([Head|Carry], [Head|Tail]) :- bezostatniego(Carry, Tail).

nakoniec(Element, [], [Element]).
nakoniec(Element, [Head|Tail], [Head|Carry]) :- nakoniec(Element, Tail, Carry).


zakazdym([], X, []).
zakazdym([Head|Tail], X, [Head, X|Carry]) :- zakazdym(Tail,X,Carry).

codrugi([], []).
codrugi([_], []).
codrugi([_, Head|Tail], [Head|Carry]) :- codrugi(Tail, Carry).

wstaw_za([X|Tail], X, Y, [X, Y|Tail]).
wstaw_za([Head|Tail], X, Y, [Head|Carry]):- wstaw_za(Tail, X, Y, Carry).

lewoprawo(X, [X|LP], _, LP).
lewoprawo(X, [Head|Tail], [Head|Carry], LP):- lewoprawo(X, Tail, Carry, LP).


wymiana(X, Y, [], []).
wymiana(X, Y, [X|Tail], [Y|Carry]):- wymiana(X, Y, Tail, Carry).
wymiana(X, Y, [Head|Tail], [Head|Carry]):- wymiana(X, Y, Tail, Carry).

polacz([], []).
polacz([Head|Tail], X):-  polacz(Tail, Res), append(Head, Res, X).
```

lab 5

```
klonuj([], []).
klonuj([Head|Tail], [[Head, Head]|Carry]) :- klonuj(Tail, Carry).


sumuj([], 0).
sumuj([Head|Tail], M) :- sumuj(Tail, X), M is X+Head.

polowki([],[],[]).
polowki([Head|Tail], [Head|L], X) :-
    append(TrimTail, [Last], Tail),
    polowki(TrimTail, L, P),
    append(P, [Last], X).


srodek([Mid], Mid).
srodek([_|Tail], Mid) :- append(TrimTail, [_], Tail), srodek(TrimTail, Mid).

ile_wiekszych([], _, 0).
ile_wiekszych([Head|Tail], X, M) :- ile_wiekszych(Tail, X, N), Head > X, M is N + 1.
ile_wiekszych([Head|Tail], X, N) :- ile_wiekszych(Tail, X, N), Head =< X.

plus_minus([], 0).
plus_minus([Head|Tail], M) :- plus_minus(Tail,N),Head = '+',  M is N + 1.
plus_minus([Head|Tail], M) :- plus_minus(Tail,N),Head = '-', M is N - 1.

wstawiaj([], X, 1, [X]).
wstawiaj([Head|Tail], X, 1, [X, Head|Tail]).
wstawiaj([Head|Tail], X, Pos, [Head|Carry]) :-
    DecPos is Pos - 1,
    wstawiaj(Tail, X, DecPos, Carry).

usuwaj([Head|Tail],1,Tail).
usuwaj([Head|Tail], N, Carry) :- NewN is N - 1,usuwaj(Tail, NewN, Carry).


rozdziel([], _, [], []).
rozdziel([Head|Tail], X, [Head|L], P) :- Head < X, rozdziel(Tail, X, L, P).
rozdziel([Head|Tail], X, L, [Head|P]) :- Head >= X, rozdziel(Tail, X, L, P).

powiel([],[],[]).
powiel([El|Tail], [Times|Tail2], List) :- powiel(Tail, Tail2, Carry), powiel1(El, Times, Reps),
append(Reps, Carry, List).

powiel1(_, 0, []).
powiel1(X, N, [X|Carry]) :- N > 0, NewN is N - 1, powiel1(X, NewN, Carry).
```

lab 6

```
najwiekszy([Last], Last).
najwiekszy([Head|Tail], Head) :- najwiekszy(Tail, Max), Head > Max.
najwiekszy([Head|Tail], Max) :- najwiekszy(Tail, Max), Head =< Max.

liniowa([_, _]).
liniowa([H1, H2, H3|Tail]) :- liniowa([H2, H3|Tail]), H1 - H2 =:= H2 - H3.


rozbij([], [], []).
rozbij(List, [Len|Tail], [List_curr|Carry]) :-
    Len > 0,
    length(List_curr, Len),
    append(List_curr, List_next, List),
    rozbij(List_next, Tail, Carry).


przynaleznosc([], [_, _], [], []).
przynaleznosc([Element|Elements_tail], [L, P], [Element|Carry], Out) :-
    Element >= L,
    Element =< P,
    przynaleznosc(Elements_tail, [L, P], Carry, Out),
    !.
przynaleznosc([Element|Elements_tail], [L, P], In, [Element|Carry]) :-
    Element > P,
    przynaleznosc(Elements_tail, [L, P], In, Carry),
    !.
przynaleznosc([Element|Elements_tail], [L, P], In, [Element|Carry]) :-
    Element < L,
    przynaleznosc(Elements_tail, [L, P], In, Carry).


decnabin(0, [0]).
decnabin(1, [1]).
decnabin(N, Bin) :-
    N>1,
    Ndiv2 is div(N, 2),
    Nmod2 is mod(N, 2),
    decnabin(Ndiv2, Carry),
    append(Carry, [Nmod2], Bin).

rozdziel([], _, [], []).
rozdziel([Head|Tail], X, [Head|L], P) :- Head < X, rozdziel(Tail, X, L, P).
rozdziel([Head|Tail], X, L, [Head|P]) :- Head >= X, rozdziel(Tail, X, L, P).

szybkisort([], []).
szybkisort([Pivot|Tail], Srt) :-
    rozdziel(Tail, Pivot, L, P),
```

```
    szybkisort(L, CarryL),
    szybkisort(P, CarryP),
    append(CarryL, [Pivot|CarryP], Srt).

doposort([], X, [X]).
doposort([Head|Tail], X, [Head|Carry]) :-
    X > Head,
    doposort(Tail, X, Carry).
doposort([Head|Tail], X, [X, Head|Tail]) :-
    X =< Head.

wstawsort([], []).
wstawsort([Head|Tail], Sorted) :-
    wstawsort(Tail, Carry),
    doposort(Carry,Head, Sorted).

% unikalne(allowed, unique).
unikalne([], []).
unikalne([A|As], [A|Us]) :-
    not(member(A, As)),
    unikalne(As, Us),
    !.
unikalne([A|As], Us) :-
    member(A, As),
    unikalne(As, Us).


% najblizszy(X, List, Y member of List that minimalise abs(X-Y))
najblizszy(_, [El], El).
najblizszy(X, [E|Es], E) :-
    najblizszy(X, Es, Y),
    abs(X-E) < abs(X-Y),
    !.
najblizszy(X, [E|Es], Y) :-
    najblizszy(X, Es, Y),
    abs(X-E) >= abs(X-Y).


%eratostenes(2=<N, Primes)
eratostenes(N, Primes) :-
    ciagliczb(2, N, Ns),
    sito(Ns,  Primes).

%ciagliczb(min inclusive, max inclusive, List)
ciagliczb(M, N, [N]) :-
    M =:= N.
ciagliczb(M, N, [M|Carry]) :-
          M < N,
```

```
        NewM is M + 1,
        ciagliczb(NewM, N, Carry).

%odsiewaj(filter mod, list, list filtred).
odsiewaj(_, [], []).
odsiewaj(Mod, [L|Ls], [L|Fs]) :-
   mod(L, Mod) =\= 0,
   odsiewaj(Mod, Ls, Fs).
odsiewaj(Mod, [L|Ls], Fs) :-
   mod(L, Mod) =:= 0,
   odsiewaj(Mod, Ls, Fs).

sito([], []).
sito([L|Ls], [L|Carry]) :-
   odsiewaj(L, Ls, Filtred),
   sito(Filtred, Carry).
```

lab 7

```
% sumuj(lista, wynik)
sumuj([], 0).
sumuj([Head|Tail], M) :- sumuj(Tail, X), M is X+Head.


%podzielniki(liczba, najmniejszy dozwolony dzielnik, lista dzielników)
podzielniki(N, N, []).
podzielniki(N, D, [D|Lcarry]) :-
    D<N,
    mod(N, D) =:= 0,
    Dn is D + 1,
    podzielniki(N, Dn, Lcarry),
    !.
podzielniki(N, D, L) :-
    D<N,
    mod(N, D) =\= 0,
    Dn is D + 1,
    podzielniki(N, Dn, L).

doskonala(N) :-
    podzielniki(N, 1, Divs),
    sumuj(Divs, Val),
    Val =:= N.


%przekroj(zbior A, zbior B, iloczyn AB)
przekroj([], _, []).
przekroj(_, [], []).
przekroj([L1|L1s], [L2|L2s], [L1|Lcarry]) :-
    L1 =:= L2,
    przekroj(L1s, L2s, Lcarry),
    !.
przekroj([L1|L1s], [L2|L2s], Lcarry) :-
    L1 < L2,
    przekroj(L1s, [L2|L2s], Lcarry),
    !.
przekroj([L1|L1s], [L2|L2s], Lcarry) :-
    L1 > L2,
    przekroj([L1|L1s], L2s, Lcarry).

glowyogony([], [], []).
glowyogony([[Head|Tail]|Ls], [Head|Lgs], [Tail|Los]) :-
    glowyogony(Ls, Lgs, Los).

slad([[A]], A).
slad(M, S) :-
    glowyogony(M, [Lg|_], [_|Los]),
```

```
    slad(Los, Scarry),
    S is Scarry + Lg.

rozszerzaj([[A, B]], [A, B]).
rozszerzaj([[A, B]|Rest], [Ac, Bc]) :-
    rozszerzaj(Rest, [Ac, Bc]),
    B=< Bc,
    A>= Ac,
    !.
rozszerzaj([[A, B]|Rest], [Ac, B]) :-
    rozszerzaj(Rest, [Ac, Bc]),
    B > Bc,
    A>= Ac,
    !.
rozszerzaj([[A, B]|Rest], [A, Bc]) :-
    rozszerzaj(Rest, [Ac, Bc]),
    B=< Bc,
    A < Ac,
    !.
rozszerzaj([[A, B]|Rest], [A, B]) :-
    rozszerzaj(Rest, [Ac, Bc]),
    B> Bc,
    A< Ac.

ciagliczb(M, N, [N]) :-
    M =:= N.
ciagliczb(M, N, [M|Carry]) :-
            M < N,
            NewM is M + 1,
            ciagliczb(NewM, N, Carry).

liczpodziel(N, N, 0).
liczpodziel(N, D, L) :-
    D<N,
    mod(N, D) =:= 0,
    Dn is D + 1,
    liczpodziel(N, Dn, Lcarry),
    L is Lcarry +1,
    !.
liczpodziel(N, D, L) :-
    D<N,
    mod(N, D) =\= 0,
    Dn is D + 1,
    liczpodziel(N, Dn, L).

antyp([], _).
antyp([Head|Tail], Max) :-
    liczpodziel(Head, 1, Divs),
```

```
    Divs < Max,
    antyp(Tail, Max).

antypierwsza(N) :-
    liczpodziel(N, 1, Divs),
    Npre is N - 1,
    ciagliczb(1, Npre, List),
    antyp(List,Divs).

rozmien(Coins, N, [Current]) :-
    append(_, [Current], Coins),
    Current =< N,
    NewN is N - Current,
          NewN =:= 0,
    !.
rozmien(Coins, N, Res) :-
    append(_, [Current], Coins),
    Current =< N,
    NewN is N - Current,
    rozmien(Coins, NewN, List),
    append(List, [Current], Res),
    !.
rozmien(Coins, N, List) :-
    append(Smaller, [Current], Coins),
    Current > N,
    rozmien(Smaller, N, List).

wpolu([],_, []).
wpolu([[X,Y]|Tail], [[Xa, Ya], [Xb, Yb]], [[X, Y]|Carry]) :-
    X>=Xa,
    X=<Xb,
    Y>=Ya,
    Y=<Yb,
    wpolu(Tail, [[Xa, Ya], [Xb, Yb]], Carry),
    !.
wpolu([[X,_]|Tail], [[Xa, Ya], [Xb, Yb]], Carry) :-
     X < Xa,
     wpolu(Tail, [[Xa, Ya], [Xb, Yb]], Carry),
     !.
wpolu([[X,_]|Tail], [[Xa, Ya], [Xb, Yb]], Carry) :-
     X > Xb,
     wpolu(Tail, [[Xa, Ya], [Xb, Yb]], Carry),
    !.
wpolu([[_,Y]|Tail], [[Xa, Ya], [Xb, Yb]], Carry) :-
     Y < Ya,
     wpolu(Tail, [[Xa, Ya], [Xb, Yb]], Carry),
     !.
wpolu([[_,Y]|Tail], [[Xa, Ya], [Xb, Yb]], Carry) :-
```

```
    Y > Yb,
    wpolu(Tail, [[Xa, Ya], [Xb, Yb]], Carry).


minX([[X, _]], X).
minX([[X, _]|Tail], X) :-
   minX(Tail, Min),
   X < Min,
   !.
minX([[X, _]|Tail], Min) :-
   minX(Tail, Min),
   X >= Min.

maxX([[X, _]], X).
maxX([[X, _]|Tail], X) :-
   maxX(Tail, Max),
   X > Max,
   !.
maxX([[X, _]|Tail], Max) :-
   maxX(Tail, Max),
   X =< Max.

minY([[_, Y]], Y).
minY([[_, Y]|Tail], Y) :-
   minY(Tail, Min),
   Y < Min,
   !.
minY([[_, Y]|Tail], Min) :-
   minY(Tail, Min),
   Y >= Min.

maxY([[_, Y]], Y).
maxY([[_, Y]|Tail], Y) :-
   maxY(Tail, Max),
   Y > Max,
   !.
maxY([[_, Y]|Tail], Max) :-
   maxY(Tail, Max),
   Y =< Max.

minpole(List, [[Xa, Ya], [Xb, Yb]]) :-
   minX(List, Xa),
   maxX(List, Xb),
   minY(List, Ya),
   maxY(List, Yb).
```

```
zamien_d([], []).
zamien_d([A], [A]).
zamien_d([A, B|L1s], [B|Carry]) :- %change
    A < B,
    zamien_d([A| L1s], Carry),
    !.
zamien_d([A, B|L1s], [A|Carry]) :-
    A >= B,
    zamien_d([B|L1s], Carry).



zamien_g([], []).
zamien_g([A], [A]).
zamien_g([A, B|L1s], [B|Carry]) :- %change
    A > B,
    zamien_g([A| L1s], Carry),
    !.
zamien_g([A, B|L1s], [A|Carry]) :-
    A =< B,
    zamien_g([B|L1s], Carry).

odwracaj([], []).
odwracaj([L|Ls], R) :-
    odwracaj(Ls, Carry),
    append(Carry, [L], R).

koktajl([], []).
koktajl([A], [A]).
koktajl(L, S) :-
    zamien_g(L, Bubbleup),
    odwracaj(Bubbleup, Bubbleup_reversed),
    zamien_d(Bubbleup_reversed, Bubbledown_reversed),
    odwracaj(Bubbledown_reversed, Bubbledown),
    append([Min|Mid], [Max], Bubbledown),
    koktajl(Mid, Sorted_mid),
    append([Min|Sorted_mid], [Max], S).


ciagFib(N,F1,_,[]) :-
 F1 > N,
 !.
ciagFib(N,F1,F2,[F1|L]) :-
 F1 =< N, F3 is F2 + F1,
 ciagFib(N,F2,F3,L),
 !.

rozklad(LF,N,[N]) :-
```

```
 append(_,[X],LF),
 N-X =:= 0,
 !.
rozklad(LF,N,L) :-
 append(_,[X],LF),
 X =< N,
 N1 is N-X,
 rozklad(LF,N1,L1),
 append(L1,[X],L),
 !.
rozklad(LF,N,L) :-
 append(LF1,[X],LF),
 X > N,
 rozklad(LF1,N,L),
 !.

rozklad_fib(N,L) :-
 ciagFib(N,0,1,L1),
 rozklad(L1,N,L).
```