```
/* LABORATORIUM 6 */


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

% Zadanie1

% Predykat var
var(X).     % true, bo X nie ma ustalnej wartości
nonvar(X).  % false, bo X nie ma ustalonej wartości
var(X),X=2. % true, bo najpierw jest wolane var(X), wtedy jeszcze
        % X nie ma wartosci
X=2,var(X).     % false bo X ma wartosc 2

% Predykat atom
?- atom(X).
false.

?- atom(3).
false.

?- atom(a).
true.

?- atom(+).
true.

?- atom(:=).
true.

?- atom('ala').
true.

% Predykat atomic
?- atomic(a).
true.

?- atomic(3).
true.

?- atomic(+).
true.

?- atomic(X).
false.

% Predykat number
?- number(ala).
false.

?- number(3).
true.

?- number(3.13).
true.

% Predykat integer
?- integer(3).
true.

?- integer(3.14).
false.

?- integer(ala).
false.

% Predykat float
?- float(3).
false.

?- float(3.14).
true.

?- float(ala).
false.

% Predykat compound
[trace]  ?- compound(ala).
false.

[trace]  ?- compound(ala(ma,kota)).
true.

[trace]  ?- compound(3).
false.

[trace]  ?- compound([]).
false.

[trace]  ?- compound([1]).
```

```
true.

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

% Predykat =..

?- A =.. [ala, ma, asa].
A = ala(ma, asa).

?- ala(ma,kota,w(ciapki(rozowe))) =.. A.
A = [ala, ma, kota, w(ciapki(rozowe))].


% Predykat functor

?- functor(ala(ma,kota),F,A).
F = ala,
A = 2.

?- CzyTo = ala, OLiczbie = 2, functor(ala(ma,kota),CzyTo,OLiczbie).
CzyTo = ala,
OLiczbie = 2.

?- CzyTo = kasia, OLiczbie = 2, functor(ala(ma,kota),CzyTo,OLiczbie).
false.

?- functor(ala(ma,kota),F,_), write('To jest funktor \''), write(F), write('\'').
To jest funktor 'ala'
F = ala.

?- functor(A, riverside, 3).
A = riverside(_G1748, _G1749, _G1750).


% Predykat arg

?- arg(X,ala_ma(kota,psa,schiza),A).
X = 1,
A = kota ;
X = 2,
A = psa ;
X = 3,
A = schiza.

?- arg(2,ala_ma(kota,psa,schiza),A).
A = psa.

?- functor(A, riverside, 4), arg(1,A,voices), arg(4,A,head).
A = riverside(voices, _G2065, _G2066, head).


a(1). a(2). b(4). b(3).

wyp0(F,_) :-
    call(F).

wyp1(F,X) :-
    F,
    F =.. [_,X].

wyp2(F,X) :-
    functor(Pred, F, 1),
    Pred,
    Pred =.. [_,X].

wyp3(F/A,X) :-
    A = 1,
    functor(Pred,F,A),
    Pred,
    Pred =.. [_,X].


]  ?- wyp0(a(X),X).
   Call: (7) wyp0(a(_G2845), _G2845) ? creep
   Call: (8) a(_G2845) ? creep
   Exit: (8) a(1) ? creep
   Exit: (7) wyp0(a(1), 1) ? creep
X = 1 ;
   Redo: (8) a(_G2845) ? creep
   Exit: (8) a(2) ? creep
   Exit: (7) wyp0(a(2), 2) ? creep
X = 2.

[trace]  ?- wyp0(b(X),X).
   Call: (7) wyp0(b(_G2845), _G2845) ? creep
   Call: (8) b(_G2845) ? creep
   Exit: (8) b(4) ? creep
   Exit: (7) wyp0(b(4), 4) ? creep
X = 4 ;
   Redo: (8) b(_G2845) ? creep
```

```
   Exit: (8) b(3) ? creep
   Exit: (7) wyp0(b(3), 3) ? creep
X = 3.

[trace]  ?- wyp1(a(_),X).
   Call: (7) wyp1(a(_G2845), _G2848) ? creep
   Call: (8) a(_G2845) ? creep
   Exit: (8) a(1) ? creep
   Call: (8) a(1)=..[_G2921, _G2848] ? creep
   Exit: (8) a(1)=..[a, 1] ? creep
   Exit: (7) wyp1(a(1), 1) ? creep
X = 1 ;
   Redo: (8) a(_G2845) ? creep
   Exit: (8) a(2) ? creep
   Call: (8) a(2)=..[_G2921, _G2848] ? creep
   Exit: (8) a(2)=..[a, 2] ? creep
   Exit: (7) wyp1(a(2), 2) ? creep
X = 2.

[trace]  ?- wyp1(b(_),X).
   Call: (7) wyp1(b(_G2845), _G2848) ? creep
   Call: (8) b(_G2845) ? creep
   Exit: (8) b(4) ? creep
   Call: (8) b(4)=..[_G2921, _G2848] ? creep
   Exit: (8) b(4)=..[b, 4] ? creep
   Exit: (7) wyp1(b(4), 4) ? creep
X = 4 ;
   Redo: (8) b(_G2845) ? creep
   Exit: (8) b(3) ? creep
   Call: (8) b(3)=..[_G2921, _G2848] ? creep
   Exit: (8) b(3)=..[b, 3] ? creep
   Exit: (7) wyp1(b(3), 3) ? creep
X = 3.


[trace]  ?- wyp2(a,X).
   Call: (7) wyp2(a, _G2846) ? creep
   Call: (8) functor(_G2921, a, 1) ? creep
   Exit: (8) functor(a(_G2914), a, 1) ? creep
   Call: (8) a(_G2914) ? creep
   Exit: (8) a(1) ? creep
   Call: (8) a(1)=..[_G2916, _G2846] ? creep
   Exit: (8) a(1)=..[a, 1] ? creep
   Exit: (7) wyp2(a, 1) ? creep
X = 1 ;
   Redo: (8) a(_G2914) ? creep
   Exit: (8) a(2) ? creep
   Call: (8) a(2)=..[_G2916, _G2846] ? creep
   Exit: (8) a(2)=..[a, 2] ? creep
   Exit: (7) wyp2(a, 2) ? creep
X = 2.


[trace]  ?- wyp2(b,X).
   Call: (7) wyp2(b, _G2846) ? creep
   Call: (8) functor(_G2921, b, 1) ? creep
   Exit: (8) functor(b(_G2914), b, 1) ? creep
   Call: (8) b(_G2914) ? creep
   Exit: (8) b(4) ? creep
   Call: (8) b(4)=..[_G2916, _G2846] ? creep
   Exit: (8) b(4)=..[b, 4] ? creep
   Exit: (7) wyp2(b, 4) ? creep
X = 4 ;
   Redo: (8) b(_G2914) ? creep
   Exit: (8) b(3) ? creep
   Call: (8) b(3)=..[_G2916, _G2846] ? creep
   Exit: (8) b(3)=..[b, 3] ? creep
   Exit: (7) wyp2(b, 3) ? creep
X = 3.


[trace]  ?- wyp3(a/1,X).
   Call: (7) wyp3(a/1, _G2879) ? creep
   Call: (8) 1=1 ? creep
   Exit: (8) 1=1 ? creep
   Call: (8) functor(_G2957, a, 1) ? creep
   Exit: (8) functor(a(_G2950), a, 1) ? creep
   Call: (8) a(_G2950) ? creep
   Exit: (8) a(1) ? creep
   Call: (8) a(1)=..[_G2952, _G2879] ? creep
   Exit: (8) a(1)=..[a, 1] ? creep
   Exit: (7) wyp3(a/1, 1) ? creep
X = 1 ;
   Redo: (8) a(_G2950) ? creep
   Exit: (8) a(2) ? creep
   Call: (8) a(2)=..[_G2952, _G2879] ? creep
   Exit: (8) a(2)=..[a, 2] ? creep
   Exit: (7) wyp3(a/1, 2) ? creep
X = 2.


[trace]  ?- wyp3(b/1,X).
```

```
[trace]  ?- wyp3(b/1,X).
   Call: (7) wyp3(b/1, _G3131) ? creep
   Call: (8) 1=1 ? creep
   Exit: (8) 1=1 ? creep
   Call: (8) functor(_G3209, b, 1) ? creep
   Exit: (8) functor(b(_G3202), b, 1) ? creep
   Call: (8) b(_G3202) ? creep
   Exit: (8) b(4) ? creep
   Call: (8) b(4)=..[_G3204, _G3131] ? creep
   Exit: (8) b(4)=..[b, 4] ? creep
   Exit: (7) wyp3(b/1, 4) ? creep
X = 4 ;
   Redo: (8) b(_G3202) ? creep
   Exit: (8) b(3) ? creep
   Call: (8) b(3)=..[_G3204, _G3131] ? creep
   Exit: (8) b(3)=..[b, 3] ? creep
   Exit: (7) wyp3(b/1, 3) ? creep
X = 3.


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



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

:- op(100, xfy, matka).
julia matka marcin.
:- op(300, xfx, ma).
:- op(200, xfy, i).
jas ma kota i psa.
ala ma jasia i angine i dosc_agh.
rybki i kanarki.


?- ma(X,Y).
X = jas,
Y = kota i psa ;
X = ala,
Y = jasia i angine i dosc_agh.

?- ma(X,i(A,B)).
X = jas,
A = kota,
B = psa ;
X = ala,
A = jasia,
B = angine i dosc_agh.

?- ma(X,i(B,i(C,D))).
X = ala,
B = jasia,
C = angine,
D = dosc_agh.

?- Kto ma Co.
Kto = jas,
Co = kota i psa ;
Kto = ala,
Co = jasia i angine i dosc_agh.

?- Kto ma Co i Coinnnego.
Kto = jas,
Co = kota,
Coinnnego = psa ;
Kto = ala,
Co = jasia,
Coinnnego = angine i dosc_agh.

?- Kto ma Co i Coinnego i Jeszcze.
Kto = ala,
Co = jasia,
Coinnego = angine,
Jeszcze = dosc_agh

?- display(jas ma kota i psa).
ma(jas,i(kota,psa))
true.

?- display(ala ma jasia i angine i dosc_agh).
ma(ala,i(jasia,i(angine,dosc_agh)))
true.

?- i(A,B).
A = rybki,
B = kanarki.



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

```
,  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  --  ,



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

:- include(readstr).

odpowiedz :-
    write('\' matka \' czy \'ojciec\'? '),
    read_atom(X),
    write('kogo ? '),
    read_atom(Y),
    Q =.. [X, Kto, Y],
    display(Q),
    call(Q),
    write(Kto), nl.

rozwiaz1(G) :- call(G).

rozwiaz2(true) :- !.
rozwiaz2((G1,G2)) :- !,
    rozwiaz2(G1),
    rozwiaz2(G2).
rozwiaz2(G) :-
    clause(G,B).
    rozwiaz2(B).

rozwiaz3(true) :- !.
rozwiaz3((G1, G2)) :- !,
    rozwiaz3(G1),
    rozwiaz3(G2).
rozwiaz3(G) :-
    write('Wywoluje: '), write(G), nl,
    clause(G,B),
    rozwiaz3(B),
    write('Wyjscie:   '), write(G), nl.




?- listing(kobieta).
kobieta(kasia).
kobieta(ania).
kobieta(basia).
kobieta(gosia).

true.

?- call(kobieta(X)).
X = kasia ;
X = ania ;
X = basia ;
X = gosia.

?- clause(kobieta(X),B).
X = kasia,
B = true ;
X = ania,
B = true ;
X = basia,
B = true ;
X = gosia,
B = true.

?- listing(matka).
matka(A, B) :-
    rodzic(A, B),
    kobieta(A).

true.

?- Kto = kasia, call(matka(Kto, Kogo)), write(Kto), write(' jest matka '), write(Kogo).
kasia jest matka robert
Kto = kasia,
Kogo = robert.

?- Matka = kasia, Dziecko = robert, clause(matka(Matka,Dziecko),Kiedy), write(Matka), write( ' jest matka '), write(Dziecko), write(
kasia jest matka robert wtedy gdy rodzic(kasia,robert),kobieta(kasia)
Matka = kasia,
Dziecko = robert,
Kiedy = (rodzic(kasia, robert), kobieta(kasia)).

?- [meta].
true.

?- [rodzina1].
true.

?- odpowiedz.
' matka ' czy 'ojciec'? ojciec
```

```
kogo ? robert
ojciec(_G1763,robert)tomek
true.

[trace]  ?- rozwiaz1(matka(kasia,X)).
   Call: (7) rozwiaz1(matka(kasia, _G1816)) ? creep
   Call: (8) matka(kasia, _G1816) ? creep
   Call: (9) rodzic(kasia, _G1816) ? creep
   Exit: (9) rodzic(kasia, robert) ? creep
   Call: (9) kobieta(kasia) ? creep
   Exit: (9) kobieta(kasia) ? creep
   Exit: (8) matka(kasia, robert) ? creep
   Exit: (7) rozwiaz1(matka(kasia, robert)) ? creep
X = robert.

[trace]  ?- rozwiaz1((matka(kasia,X), matka(Y,robert))).
   Call: (7) rozwiaz1((matka(kasia, _G1816), matka(_G1818, robert))) ? creep
   Call: (9) matka(kasia, _G1816) ? creep
   Call: (10) rodzic(kasia, _G1816) ? creep
   Exit: (10) rodzic(kasia, robert) ? creep
   Call: (10) kobieta(kasia) ? creep
   Exit: (10) kobieta(kasia) ? creep
   Exit: (9) matka(kasia, robert) ? creep
   Call: (9) matka(_G1818, robert) ? creep
   Call: (10) rodzic(_G1818, robert) ? creep
   Exit: (10) rodzic(kasia, robert) ? creep
   Call: (10) kobieta(kasia) ? creep
   Exit: (10) kobieta(kasia) ? creep
   Exit: (9) matka(kasia, robert) ? creep
   Exit: (7) rozwiaz1((matka(kasia, robert), matka(kasia, robert))) ? creep
X = robert,
Y = kasia ;
   Redo: (10) rodzic(_G1818, robert) ? creep
   Exit: (10) rodzic(tomek, robert) ? creep
   Call: (10) kobieta(tomek) ? creep
   Fail: (10) kobieta(tomek) ? creep
   Fail: (9) matka(_G1818, robert) ? creep
   Fail: (7) rozwiaz1((matka(kasia, _G1816), matka(_G1818, robert))) ? creep
false.


[trace]  ?- rozwiaz2((matka(kasia,X), matka(Y,robert))).
   Call: (7) rozwiaz2((matka(kasia, _G1846), matka(_G1848, robert))) ? creep
   Call: (8) rozwiaz2(matka(kasia, _G1846)) ? creep
^  Call: (9) clause(matka(kasia, _G1846), _G1953) ? creep
^  Exit: (9) clause(matka(kasia, _G1846), (rodzic(kasia, _G1846), kobieta(kasia))) ? creep
   Exit: (8) rozwiaz2(matka(kasia, _G1846)) ? creep
   Call: (8) rozwiaz2(matka(_G1848, robert)) ? creep
^  Call: (9) clause(matka(_G1848, robert), _G1967) ? creep
^  Exit: (9) clause(matka(_G1848, robert), (rodzic(_G1848, robert), kobieta(_G1848))) ? creep
   Exit: (8) rozwiaz2(matka(_G1848, robert)) ? creep
   Exit: (7) rozwiaz2((matka(kasia, _G1846), matka(_G1848, robert))) ? creep
true .

?- rozwiaz3((matka(kasia,X), matka(Y,robert))).
Wywoluje: matka(kasia,_G2271)
Wywoluje: rodzic(kasia,_G2271)
Wyjscie:   rodzic(kasia,robert)
Wywoluje: kobieta(kasia)
Wyjscie:   kobieta(kasia)
Wyjscie:   matka(kasia,robert)
Wywoluje: matka(_G2273,robert)
Wywoluje: rodzic(_G2273,robert)
Wyjscie:   rodzic(kasia,robert)
Wywoluje: kobieta(kasia)
Wyjscie:   kobieta(kasia)
Wyjscie:   matka(kasia,robert)
X = robert,
Y = kasia ;
Wyjscie:   rodzic(tomek,robert)
Wywoluje: kobieta(tomek)
false.
```

```
/*
 * rodzina1.pl
 * GJN 2005,6
 * based on Bratko, 3rd ed, ch.1, p.17
 *
 */

rodzic(kasia,robert).
rodzic(tomek,robert).
rodzic(tomek,ania).
rodzic(robert,gosia).
rodzic(robert,basia).
rodzic(basia,janek).
```

```
kobieta(kasia).
kobieta(ania).
kobieta(basia).
kobieta(gosia).

mezczyzna(tomek).
mezczyzna(robert).
mezczyzna(janek).

dziecko(X,Y) :-
    rodzic(Y,X).

matka(X,Y) :-
    rodzic(X,Y),
    kobieta(X).

ojciec(X,Y) :-
    rodzic(X,Y),
    mezczyzna(X).

dziadkowie(X,Y) :-
    rodzic(X,Z),
    rodzic(Z,Y).

dziadek(X,Y) :-
    dziadkowie(X,Y),
    mezczyzna(X).

babcia(X,Y) :-
    dziadkowie(X,Y),
    kobieta(X).

siostra(X,Y) :-
    rodzic(Z,X),
    rodzic(Z,Y),
    kobieta(X),
    X \= Y.

brat(X,Y) :-
    rodzic(Z,X),
    rodzic(Z,Y),
    mezczyzna(X),
    X \= Y.

przodek(X,Y) :-
    rodzic(X,Y).

przodek(X,Z) :-
    rodzic(X,Y),
    przodek(Y,Z).

potomek(X,Y) :-
    rodzic(Y,X).

potomek(Z,X) :-
    rodzic(Y,Z),
    potomek(Y,X).

%%
ma_dziecko(X) :-
    rodzic(X,_).

%% ex 1.3
szczesliwy(X) :-
    ma_dziecko(X).

ma_dwoje_dzieci(X) :-
    rodzic(X,Y),
    siostra(_,Y).

ma_dwoje_dzieci(X) :-
    rodzic(X,Y),
    brat(_,Y).

%% ex 1.4
wnuk(X,Z) :-
    rodzic(Y,X),
    rodzic(Z,Y).

%% ex 1.5
ciocia(X,Y) :-
    rodzic(Z,Y),
    siostra(X,Z).




% 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 READSTR.PL
% Reading and writing lines of text

% Uses get0, and works in almost all Prologs (not Arity).


% read_str(-String)
%   Accepts a whole line of input as a string (list of ASCII codes).
%   Assumes that the keyboard is buffered.

read_str(String) :- get0(Char),
                    read_str_aux(Char,String).

read_str_aux(-1,[]) :- !.    % end of file
read_str_aux(10,[]) :- !.    % end of line (UNIX)
read_str_aux(13,[]) :- !.    % end of line (DOS)

read_str_aux(Char,[Char|Rest]) :- read_str(Rest).


% read_atom(-Atom)
%  Reads a line of input and converts it to an atom.
%  See text concerning name/2 vs. atom_codes/2.

read_atom(Atom) :-
   read_str(String),
   name(Atom,String).    % or preferably atom_codes(Atom,String).


% read_num(-Number)
%  Reads a line of input and converts it to a number.
%  See text concerning name/2 vs. number_codes/2.

read_num(Atom) :-
   read_str(String),
   name(Atom,String).    % or preferably number_codes(Atom,String).


% write_str(+String)
%  Outputs the characters corresponding to a list of ASCII codes.

write_str([Code|Rest]) :- put(Code), write_str(Rest).
write_str([]).



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