```
%
%                   Deklaratív Programozás gyakorlat
%               Prolog programozás: meta-logikai eljárások

%

% 1.  Atomok szeletelése

%     Egy A atom prefixumának nevezünk egy P atomot, ha P az A első
%     valahány karakterét tartalmazza, az A-beli sorrend megtartásával.

%     % atom_prefix(+Atom, ?Prefix, +N): Atom-nak Prefix N hosszú prefixuma.
%     % Másszóval: az Atom első N karakteréből képzett névkonstans a Prefix atom.

%     | ?- atom_prefix(abcde, Prefix, 0).
%     Prefix = '' ? ; no
%     | ?- atom_prefix(abcde, Prefix, 3).
%     Prefix = abc ? ; no
%     | ?- atom_prefix(abcde, Prefix, 5).
%     Prefix = abcde ? ; no
%     | ?- atom_prefix(abcde, Prefix, 6).
%     no

%     Nem használhatja a sub_atom/5 beépített eljárást!
%     Ötlet: használja az atom_codes és prefix_length eljárásokat.

:- use_module(library(lists)).

atom_prefix(Atom, Prefix, N) :-
        atom_codes(Atom, Codes),
        prefix_length(Codes, Prefix_Codes, N),
        atom_codes(Prefix, Prefix_Codes).




% 2.  Általános Prolog kifejezés részkifejezéseinek vizsgálata
%
%     % mern(+K, +N): A K általános Prolog kifejezésben előforduló összes
%     % egész szám határozottan nagyobb mint N
%     % (mern = minden egész részkifejezése nagyobb mint)
%
%     | ?- mern(1, 1).
%     no
%     | ?- mern(1, 0).
%     yes
%     | ?- mern(0.0, 1).
%     yes
%     | ?- mern(f(X,[1,3,b],g(2,1,3)), 0).
%     yes
%     | ?- mern(f(X,[1,3,b],g(2,1,3)), 1).
%     no
%
% Megjegyzések:
%
%     a. A "K1 Prolog kifejezésben előfordul a K2 kifejezés" relációt
%        reflexívnek tekintjük, azaz egy K kifejezésben önmaga mindenképpen
%        előfordul. Ez a megjegyzés vonatkozik az ezután következő
%        feladatokra is.
%     b. Vigyázzon arra, hogy a kifejezésben változók is előfordulhatnak.

mern(K, N) :-
        (   integer(K) ->
            K > N
        ;   compound(K) ->
            K =.. [_Fun|Args],
            mern_lista(Args, N)
        ;   true
        ).
```

```
mern_rossz(K, N) :-
        (   integer(K) ->
            K > N
        ;   compound(K) ->
            K =.. [_Fun|Args],
            mern_rossz(Args, N)
        ;   true
        ).

% | ?- trace, mern_rossz(f(1), 0).
% % The debugger will first creep -- showing everything (trace)
%         1      1 Call: mern_rossz(f(1),0) ?
%         2      2 Call: mern_rossz([1],0) ?
%         3      3 Call: mern_rossz([1,[]],0) ?
%         4      4 Call: mern_rossz([1,[[]]],0) ?
%         5      5 Call: mern_rossz([1,[[[]]]],0) ?
%         6      6 Call: mern_rossz([1,[[[[]]]]],0) ?
%         7      7 Call: mern_rossz([1,[[[[[]]]]]],0) ?
%         8      8 Call: mern_rossz([1,[[[[[[]]]]]]],0) ?
%         9      9 Call: mern_rossz([1,[[[[[[[[]]]]]]]],0) ?
%        10     10 Call: mern_rossz([1,[[[[[[[[[]]]]]]]]],0) ?
%        11     11 Call: mern_rossz([1,[[[[[[[[...]]]]]]]],0) ?
%        12     12 Call: mern_rossz([1,[[[[[[[[...]]]]]]]],0) ?
%        13     13 Call: mern_rossz([1,[[[[[[[[...]]]]]]]],0) ?
%        14     14 Call: mern_rossz([1,[[[[[[[[...]]]]]]]],0) ?


% mern_lista(+Ks,+N): a Ks listában szereplo minden kifejezésben minden
% egész nagyobb, mint N.
mern_lista([], _N).
mern_lista([K|Ks], N) :-
        mern(K,N),
        mern_lista(Ks, N).

% mern_lista2(+L,+N): az L listának nincs olyan X eleme, amelyre nem teljesül
% hogy X-ben szereplő minden egész nagyobb mint N.
mern_lista2(L, N) :-
        \+ (   member(X, L), \+ mern(X, N)
           ).


% 3.  Általános Prolog kifejezés bizonyos részkifejezéseinek felsorolása
%
%     % reszatom(+K, ?A): A a K általános Prolog kifejezésben
%     % előforduló atom.
%
%     | ?- reszatom(a, X).
%     X = a ? ;
%     no
%     | ?- reszatom(f(X,[1,3,b],g(2,1,a0)), A).
%     A = b ? ;
%     A = [] ? ;
%     A = a0 ? ;
%     no
%
%     Megjegyzés: a struktúranevet nem tekintjük a struktúrakifejezés
%     részének.

reszatom0(K, A) :-
        (   atom(K) ->
            K = A
        ;   compound(K) ->
            K =.. [_Fun|Args],
            lista_reszatom(Args, A)
        ).

% Lista_reszatom(+L, ?A): A az L lista egy elemében előforduló atom.
lista_reszatom(Args, A) :-
        member(Arg, Args),
```

```
        reszatom0(Arg, A).


reszatom_rossz(K, A) :-
        (   atom(K) ->
            K = A
        ;   compound(K) ->
            K =.. [_Fun|Args],
            reszatom_rossz(Args, A)
        ).

% | ?- trace, reszatom_rossz(f(a), A).
% % The debugger will first creep -- showing everything (trace)
%         1      1 Call: reszatom_rossz(f(a),_933) ?
%         2      2 Call: reszatom_rossz([a],_933) ?
%         3      3 Call: reszatom_rossz([a,[]],_933) ?
%         4      4 Call: reszatom_rossz([a,[[]]],_933) ?
%         5      5 Call: reszatom_rossz([a,[[[]]]],_933) ?
%         6      6 Call: reszatom_rossz([a,[[[[]]]]],_933) ?
%         7      7 Call: reszatom_rossz([a,[[[[[]]]]]],_933) ?
%         8      8 Call: reszatom_rossz([a,[[[[[[]]]]]]],_933) ?
%         9      9 Call: reszatom_rossz([a,[[[[[[[]]]]]]]],_933) ?
%        10     10 Call: reszatom_rossz([a,[[[[[[[[]]]]]]]]],_933) ?
%        11     11 Call: reszatom_rossz([a,[[[[[[[[...]]]]]]]]],_933) ?
%        12     12 Call: reszatom_rossz([a,[[[[[[[[...]]]]]]]]],_933) ?
%        13     13 Call: reszatom_rossz([a,[[[[[[[[...]]]]]]]]],_933) ?

reszatom(K, A) :-
        (   atom(K) ->
            K = A
        ;   compound(K) ->
            K =.. [_Fun|Args],
            member(Arg, Args),
            reszatom(Arg, A)
        ).




% 4.  Általános Prolog kifejezés bizonyos részkifejezéseinek akkumulálása
%
%     % osszege(+K, ?Ossz): Ossz a K kifejezésben előforduló egész számok
%     % összege.
%
%     | ?- osszege(a, S).
%     S = 0 ? ;
%     no
%     | ?- osszege(1, S).
%     S = 1 ? ;
%     no
%     | ?- osszege(f(X,[1,3,b],g(2,1,a0)), S).
%     S = 7 ? ;
%     no

osszege(K, Sum) :-
        osszege(K, 0, Sum).

% a K kifejezésben előforduló egész számok összege + Sum0 = Sum.
osszege(K, Sum0, Sum) :-
        (   integer(K) ->
            Sum = Sum0+K
        ;   compound(K) ->
            K =.. [_Fun|Args],
            osszege_lista(Args, Sum0, Sum)
        ;   Sum = Sum0
        ).

% osszege_lista(+KL, +Ossz0, ?Ossz): A KL listában előforduló egész számok
% összege plusz az Ossz0 szám egyenlő az Ossz számmal.
```

```
osszege_lista([], Sum, Sum).
osszege_lista([X0|L0], Sum0, Sum) :-
        osszege(X0, Sum1),
        Sum2 is Sum0+Sum1,
        osszege_lista(L0, Sum2, Sum).


helyettesitese(K, HL, E) :-
        (   number(K) -> E = K
        ;   atom(K),
            (   memberchk(K-E, HL) -> true
            ;   E = 0
            )
        ).

helyettesitese2(K, _, K) :-
        number(K).
helyettesitese2(K, [M-Sz|HL], E) :-
        atom(K),
        (   K = M -> E = Sz
        ;   helyettesitese2(K, HL, E)
        ).
helyettesitese2(K, [], 0) :-
        atom(K).


erteke(K, HL, E) :-
        (   atomic(K) -> helyettesitese(K, HL, E)
        ;   K =.. [Op,A1] ->
            erteke(A1, HL, A1E),
            EKif =.. [Op,A1E],
            E is EKif
        ;   K =.. [Op,A1,A2] ->
            erteke(A1, HL, A1E),
            erteke(A2, HL, A2E),
            EKif =.. [Op,A1E,A2E],
            E is EKif
        ).


erteke2(K, HL, E) :-
        (   atom(K) ->
            (   memberchk(K-V, HL) -> E = V
            ;   E = 0
            )
        ;   number(K) -> E = K
        ;   K =.. [Op,A1] ->
            erteke2(A1, HL, A1E),
            EKif =.. [Op,A1E],
            E is EKif
        ;   K =.. [Op,A1,A2] ->
            erteke2(A1, HL, A1E),
            erteke2(A2, HL, A2E),
            EKif =.. [Op,A1E,A2E],
            E is EKif
        ).
```