```
%
%                    Deklaratív Programozás gyakorlat
%                Prolog programozás: meta-logikai eljárások
%                             2015.10.19 és 22
%


% 1.  Atomok szeletelése
%
%     Egy A atom szuffixumának nevezünk egy S atomot, ha S az A utolsó
%     valahány karakterét tartalmazza, az A-beli sorrend megtartásával.
%
%     % atom_suffix(+Atom, ?Suffix, +Before): Az Atom névkonstansból a Suffix
%     % atom úgy áll elő, hogy A elejéről elhagyunk Before számú karaktert.
%     | ?- atom_suffix(abcde, Suffix, 3).
%     Suffix = de ? ;
%     no
%     | ?- atom_suffix(abcde, Suffix, 5).
%     Suffix = '' ? ;
%     no
%     | ?- atom_suffix(abcde, Suffix, 6).
%     no
%
%     (Nem használhatja a sub_atom/5 beépített eljárást.)


atom_suffix(Atom, Suffix, Before) :-
        atom_codes(Atom, Codes),
        drop(Before, Codes, Suffix_Codes),
        atom_codes(Suffix, Suffix_Codes).

% drop(+N, +L0, -L): Az L listát úgy kapjuk, hogy az L0 lista első N elemét
% elhagyjuk, ahol N nem-negatív egész.
drop(0, L, L).
drop(N, [_X|L0], L) :-
        N > 0, N1 is N-1,
        drop(N1, L0, L).


% 2.  Szimbolikus aritmetikai kifejezések kiértékelése
%
%     Egy Prolog kifejezést nevezzünk szimbolikus aritmetikai kifejezésnek,
%     ha az az 'x' névkonstansból és számokból az is/2 beépített eljárás
%     által megengedett nulla-, egy- és kétargumentumú
%     struktúrakifejezésekből épül fel.
%
%     Példák szimbolikus aritmetikai kifejezésekre:
%
%     -x+3*(x/2+1/x)
%     max(x/exp(pi*x/2), 10)
%
%     A fenti példákban a pi egy 0-argumentumú, a a - és az exp
%     1-argumentumú, a +, /, max stb. 2-argumentumú olyan függvénykifejezés,
%     amelyet az is/2 ill. az aritmetikai összehasonlító eljárások hajlandók
%     kiérékelni (természetesen csak akkor, ha a kifejezés számokból épül
%     fel, ami a szimbolikus aritmetikai kifejezésekre nem teljesül, hiszen
%     azok az 'x' névkonstanst is tartalmazhatják).
%
%     Az alábbi feladat megoldásában feltételezheti (azaz nem kell
%     ellenőriznie), hogy a szimbolikus kifejezésben szereplő minden (x-től
%     különböző) atom, valamint 1- és 2-argumentumú struktúra olyan, amit az
%     is/2 beépített eljárás feldolgozni képes.
%
%     % erteke(+Kif, +X, ?E): A Kif szimbolikus aritmetikai kifejezés x=X
%     % helyen vett értéke E.
%
%     | ?- erteke((x+1)*x+x+2*(x+x+3), 2, E).
%     E = 22 ? ;
%     no
%     | ?- erteke(100**min(x,1/x), 2, E).
%     E = 10.0 ? ;
%     no
%     | ?- erteke(-x+3*(x/2+1/x), 1, E).
%     E = 3.5 ? ;
%     no
%     | ?- erteke(max(x*exp(pi*x/2), 10), 2, E).
% '   E = 46.281385265558534 ? ;
%     no
%     | ?- erteke(max(x*exp(pi*x/2), 10), 1, E).
%     E = 10.0 ? ;
%     no
%
%


erteke(x, X, E) :- !,
        E = X.
erteke(Kif, _, E) :-
        number(Kif), E = Kif.
erteke(Kif, X, E) :-
        Kif =.. [Fun,K1,K2],
        erteke(K1, X, E1),
        erteke(K2, X, E2),
        EKif =.. [Fun,E1,E2],
        E is EKif.
erteke(Kif, X, E) :-
        Kif =.. [Fun,K1],
        erteke(K1, X, E1),
        EKif =.. [Fun,E1],
        E is EKif.
erteke(Kif, _X, E) :-
        atom(Kif),
        % X \== x,
        E is Kif.


% 3.  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_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).


% 4.  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.

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


% 5.  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).


% 6.  Általános Prolog kifejezés bizonyos részkifejezéseinek átalakítása
%
%     % roviditett(+Kif0, ?Kif): Kif a Kif0 általános Prolog kifejezésből úgy
%     % áll elő, hogy minden, benne argumentumként előforduló nem-0-hosszú
%     % atom első karakterét elhagyjuk. (Az "argumentumként előforduló"
%     % kifejezés arra utal, hogy  a struktúraneveket változatlanul kell
%     % hagyni).
%
%     | ?- roviditett(abc, Kif).
%     Kif = bc ? ;
%     no
%     | ?- roviditett(f(ab, X, b, gh(uv), ''), Kif).
%     Kif = f(b,X,'',gh(v),'').
%     no
%     | ?- _Kif0=[abc], roviditett(_Kif0, Kif),
%          write_canonical(_Kif0), nl, write_canonical(Kif).
%     '.'(abc,[])
%     '.'(bc,']')
%     Kif = [bc|']'] ? ;
%     no

roviditett(Kif0, Kif) :-
        (   atom(Kif0), Kif0 \== '' ->
            atom_suffix(Kif0, Kif, 1)
        ;   compound(Kif0) ->
            Kif0 =.. [Fun|Args0],
            roviditett_lista(Args0, Args),
            Kif =.. [Fun|Args]
        ;   Kif = Kif0
        ).

% roviditett_lista(+L0, ?L): L az L0 listából úgy áll elő, hogy minden, az L
% lista elemeiben argumentumként előforduló nem-üres atom első
% karakterét elhagyjuk.
roviditett_lista([], []).
roviditett_lista([X0|L0], [X|L]) :-
        roviditett(X0, X),
        roviditett_lista(L0, L).
```