P2 toate a)-urile

```
%să se șteargă toate secvențele consecutive
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
add_f([], E, [E]).
add_f([H|T], E, [H|R]):- add_f(T, E, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%elim(L:list, C:list, R:list, LAST:int).

elim([], C, C, _).

elim([ A, B| T], C, R, _):-
      A =:= B - 1,
      !,
      elim([B|T], C, R, A).

elim([ A| T], C, R, LAST):-
      A =:= LAST + 1,
      !,
      elim(T, C, R, A).

elim([H|T], C, R, _):-
      add_f(C, H, CC),
      elim(T, CC, R, H).

eliminare(L, R):- elim(L, [], R, -1).
```

```
%Sa se adauge după fiecare element dintr-o lista divizorii
elementului.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
add_f([], E, [E]).
add_f([H|T], E, [H|R]):- add_f(T, E, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%add_div_o(L:list, N:int, D:int, R:list)
%adaugă în lista L toți divizorii lui N
%D divizorii posibili
%R rezultat

add_div_o(R, N, NN, R):- NN is N+1.

add_div_o(L, N, D, R):-
      N mod D =:= 0,
      !,
      add_f(L, D, LL),
      DD is D+1,
      add_div_o(LL, N, DD, R).

add_div_o(L, N, D, R):-
      DD is D+1,
      add_div_o(L, N, DD, R).

%add_div(L:list, N:int, R:list)
%adaugă la lista L toți divizorii lui N
%rezultat în R
%iio iii determinist
add_div(L, N, R):- add_div_o(L, N, 1, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%adauga(L:list, C:list, R:list)
%adaugă după fiecare element din L toți divizorii lui
%C colectoare
%R rezultat
%iio iii determinist

adauga([], R, R).

adauga([H|T], C, R):-
      add_f(C, H, CC),
      add_div(CC, H, CCC),
      adauga(T, CCC, R).
```

```
%run(L:list)
%afișează toate elementele din L urmate de divizorii lor
run(L):-
      adauga(L, [], R),
      print(R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Sa se inlocuiasca toate aparițiile unui element dintr-o lista cu un
alt
%element.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
add_f([], E, [E]).
add_f([H|T], E, [H|R]):-
      add_f(T, E, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%substituie(L:list, C:list, R:list, A:int, B:int)
%substituie orice element A din L cu elementul B
%C colectoare
%R rezultat
%iioii iiiii determinist

substituie([], C, C, _, _).

substituie([A|T], C, R, A, B):-
      !,
      add_f(C, B, CPP),
      substituie(T, CPP, R, A, B).

substituie([H|T], C, R, A, B):-
      add_f(C, H, CPP),
      substituie(T, CPP, R, A, B).

%wrapper run(L:list, R:list, A:int, B:int)
%ioii iiii determinist

run(L, R, A, B):- substituie(L, [], R, A, B).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%să se adauge in lista după 1-ul element
%al 3-lea element............ o valoare dată e
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
add_f([], E, [E]).
add_f([H|T], E, [H|R]):- add_f(T, E, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%inser(L:list, C:list, R:list, E:int, I:int, P:int)
%L lista inițială
%C colectoarea
%R rezultatul
%E elementul de adaugat
% I indexul 1 .. 2 ..
% P poziția de adăugare 1...3....7....15....
%P(n) = 2*P(n-1) + 1
%P(0) = 1
% P <- 2*I + 1; I <-- 1 inițial
inser([], C, C, _, _, _).

inser([H|T], C, R, E, I, P):-
      I =:= P,
      !,
      add_f(C, H, CC),
      add_f(CC, E, CCC),
      IPP is I+1,
      PP is I + I + 1,
      inser(T, CCC, R, E, IPP, PP).

inser([H|T], C, R, E, I, P):-
      add_f(C, H, CC),
      IPP is I+1,
      inser(T, CC, R, E, IPP, P).

inserare(L, E):-
      inser(L, [], R, E, 1, 1),
      print(R).
```

```
%intersectare fără păstrarea dublurilor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%adaugaFin(L:list, E:int, O:list)
%Adaugă elementul E la finalul Listei Listei
%L - lista inițială
%E - elementul de adaugat
%O - lista rezultat
%modele de flux: iio, iii

adaugaFin([], E, [E]).

adaugaFin([H|T], E, [H|R]):- adaugaFin(T, E, R).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%interclasare(A:list, B:list, C:list, R:list)
%interclaseaza listele A și B fără duplicate
%A, B - listele de interclasat
%C - lista colectoare
%R - rezultatul final
%modele de flux: iio, iii
%determinist


interclasare([], [], C, C).

interclasare([], [H|T], C, R):-
    last(C, U),
    H == U,
    !,
    interclasare([], T, C, R).

interclasare([H|T], [], C, R):-
    last(C, U),
    H == U,
    !,
    interclasare(T, [], C, R).

interclasare([], [H|T], C, R):-
    adaugaFin(C, H, C1),
    interclasare([], T, C1, R).

interclasare([H|T], [], C, R):-
    adaugaFin(C, H, C1),
    interclasare(T, [], C1, R).
```

```prolog
interclasare([H|T], B, C, R):-
    last(C, U),
    H == U,
    !,
    interclasare(T, B, C, R).


interclasare(A, [H|T], C, R):-
    last(C, U),
    H == U,
    !,
    interclasare(A, T, C, R).


interclasare([A1|A], [B1|B], C, R):-
    A1 < B1,
    adaugaFin(C, A1, C1),
    interclasare(A, [B1|B], C1, R).


interclasare([A1|A], [B1|B], C, R):-
    A1 == B1,
    adaugaFin(C, A1, C1),
    interclasare(A, B, C1, R).


interclasare([A1|A], [B1|B], C, R):-
    A1 > B1,
    adaugaFin(C, B1, C1),
    interclasare([A1|A], B, C1, R).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%interclasareFull(A:list, B:list, C:list)
%A, B liste de interclasat
%C: lista rezultat(interclasarea)
%determinist


interclasareFull(A, B, C):- interclasare(A, B, [], C).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%să se determine poziția elementului maxim
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
add_f([], E, [E]).
add_f([H|T], E, [H|R]):-
      add_f(T, E, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%maxim_o(L:list, M:int, R:int)

maxim_o([], M, M).
maxim_o([H|T], M, R):-
      H > M,
      !,
      maxim_o(T, H, R).

maxim_o([_|T], M, R):-
      maxim_o(T, M, R).

%wrapper
maxim([H|T], M):- maxim_o([H|T], H, M).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%poz_o(L:list, C:list, R:list, P:int, E:int)
%R lista cu pozițiile pe care se află elementul E din L
%C colectoare
%pozitia actuala din lista
%determinist iio iii

poz_o([], C, C, _, _).

poz_o([H|T], C, R, P, E):-
      H == E,
      add_f(C, P, CC),
      PP is P+1,
      poz_o(T, CC, R, PP, E).

poz_o([_|T], C, R, P, E):-
      PP is P+1,
      poz_o(T, C, R, PP, E).

%%%wrapper poz(L:list, R:list)
%R lista cu pozitiile pe care se află elementul maxim din L
%ii io determinist

poz(L, R):-
      maxim(L, E),
      poz_o(L, [], R, 1, E).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%determina predecesorul nr reprezentat pe lista
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
add_f([], E, [E]).
add_f([H|T], E, [H|R]):-
      add_f(T, E, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%pred(N:list, C:list, R:list, TR:int, E:int)
%scade 1 din N
%N număr reprezentat invers pe lista
%colectoare C
%R rezultat
%TR transportul la diferența cu 1
%E prima dată e 1, apoi se mută la 0 pe toată execuția.
%iio iii determinist

pred([],C, R, 1, _):-!,
      add_f(C, 1, R).

pred([], C, C, 0, _).

pred([H|T], C, R, TR, E):-
      DIF is H - E - TR,
      DIF < 0,
      !,
      DIFF is 10 + H - E - TR,
      add_f(C, DIFF, CC),
      pred(T, CC, R, 1, 0).

pred([H|T], C, R, TR, E):-
      DIF is H - E - TR,
      add_f(C, DIF, CC),
      pred(T, CC, R, 0, 0).

%predecesor(L:list, R:list)
%ii io determinist
predecesor(L, R):-
      reverse(L, LL),
      pred(LL, [], RR, 0, 1),
      reverse(R, RR).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Se da o lista de numere intregi.
%Se cere sa se scrie de 2 ori in lista
%fiecare număr prim.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%prim_o(N:int, D:int)
%verifică dacă N este prim
%D parcurge toți posibilii divizori între 2 și N div 2
%ii determinist

prim_o(N, D):-
      N > 1,
      JN is N div 2,
      D > JN.

prim_o(N, D):-
      N > 1,
      M is N mod D,
      M =\= 0,
      DD is D+1,
      prim_o(N, DD).

%wrapper prim(N:int)
%verifică dacă N este prim
%i determ

prim(N):- prim_o(N, 2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
add_f([], E, [E]).
add_f([H|T], E, [H|R]):-
      add_f(T, E, R).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%scrie(L:list, C:list, R:list)
%scrie de două ori toate primele din L
%C colectoare
%R rezultat
%iio iii determinist

scrie([], C, C).

scrie([H|T], C, R):-
      prim(H),
      !,
      add_f(C, H, CC),
      add_f(CC, H, CCC),
      scrie(T, CCC, R).

scrie([H|T], C, R):-
      add_f(C, H, CC),
      scrie(T, CC, R).

%wrapper run(L:lista)
%i determinist
%afișează rezultatul
run(L):-
      scrie(L, [], R),
      print(R).
```

```
%să se scrie un predicat care determina
%produsul unui nr reprezentat pe lista
%cu o anumită cifră
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%reverse_o(L:list, C:list, R:rezultat)

reverse_o([], C, C).

reverse_o([H|T], C, R):-
      reverse_o(T, [H|C], R).

reverse(L, R):- reverse_o(L, [], R).
```

```prolog
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%produs(N:list, E:int, R:int, T:transport, P:int, RR:int)
%N nr reprezentat INVERS pe lista
%N * E
%R rezultat
%T transport
%P puterea
%RR rezultat final

produs([], _, R, T, P, RR):-
      RR is (T * P) + R.

produs([H|T], E, R, TR, P, RR):-
      PROD is ( E * H ) + TR,
      PROD > 10,
      !,
      PP is PROD mod 10,
      TRN is PROD div 10,
      PRODF is R + (PP * P),
      PN is P * 10,
      produs(T, E, PRODF, TRN, PN, RR).

produs([H|T], E, R, TR, P, RR):-
      PROD is (E * H) + TR,
      PN is P * 10,
      PRODF is R + (PROD * P),
      produs(T, E, PRODF, 0, PN, RR).

%wrapper
%inversează lista de numere înainte de apel
%iii iio determinist

produs_f(L, E, R):-
      reverse(L, LL).
      produs(LL, E, 0, 0, 1, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Sa se determine cea mai lungă secvența de numere pare consecutive
dintr-o
%lista (dacă sunt mai multe secvențe de lungime maximă, una dintre
ele).
%%%%%%%%%%%%%%%%%%%%%%%%%%
add_f([], E, [E]).
add_f([H|T], E, [H|R]):- add_f(T, E, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%
%secv_o(L:list, I:int S:int, SM:int, DM:int, RS:int, RD:int)

%determină în RS și RD începutul și finalul celei mai lungi secvențe
%     de nr pare consecutive din L
%I indicele curent
%S > 0 dacă suntem într-o secvență (indică începutul ei)
%SM și DM limitele secvenței maxime găsite până la acel punct
%NU funcționează dacă secvența se termină cu ultimul element
%     pentru asta se adaugă artificial un 1 la finalul listei

%terminare lista
secv_o([], _,  _, SM, DM, SM, DM).

%începe o secvență nouă
secv_o([A,B|T], I, -1, SM, DM, RS, RD):-
      A mod 2 =:= 0,
      B =:= A + 2,
      !,
      S is I,
      IPP is I+1,
%     write('ÎNCEPE '),
%     print(A),
%     write('\n'),
      secv_o([B|T], IPP, S, SM, DM, RS, RD).


%continua o secventa
secv_o([A,B|T], I, S, SM, DM, RS, RD):-
      S =\= -1,
      A mod 2 =:= 0,
      B =:= A + 2,
      !,
%     write('CONTINUA '),
%     print(A),
%     write('\n'),
      IPP is I+1,
      secv_o([B|T], IPP, S, SM, DM, RS, RD).
```

```
%se termină o secvență care e mai lungă decât maximă
secv_o([A, B|T], I, S, SM, DM, RS, RD):-
      S =\= -1,
      A mod 2 =:= 0,
      B =\= A + 2,
      LEN is I - S,
      LENV is DM - SM,
      LEN >= LENV,
      !,
%     write('STOP >='),
%     print(A),
%     write('\n'),
      IPP is I+1,
      secv_o([B|T], IPP, -1, S, I, RS, RD).

%se termină o secvență care NU e mai lungă decât maximă
secv_o([A, B|T], I, S, SM, DM, RS, RD):-
      S =\= -1,
      A mod 2 =:= 0,
      B =\= A + 2,
      LEN is I - S,
      LENV is DM - SM,
      LEN < LENV,
      !,
      IPP is I+1,
%     write('STOP <'),
%     print(A),
%     write('\n'),
      secv_o([B|T], IPP, -1, SM, DM, RS, RD).

%în afara oricărei secvențe
secv_o([A, B|T], I, _, SM, DM, RS, RD):-
      A =\= B - 2,
      IPP is I+1,
%     write('AFARA '),
%     print(A),
%     write('\n'),
      secv_o([B|T], IPP, -1, SM, DM, RS, RD).
```

```
%în afara oricărei secvențe ultimul element
secv_o([A|T], I, _, SM, DM, RS, RD):-
      A mod 2 =:= 1,
      IPP is I+1,
%     write('AFARA '),
%     print(A),
%     write('\n'),
      secv_o(T, IPP, -1, SM, DM, RS, RD).


%secv(L:list, S:int, D:int)
%secvența maximă de pare consecutive
%începutul secvenței în S și finalul în D
%0 0 dacă nu s-a găsit așa ceva

secv(L, S, D):-
      add_f(L, 1, LL),
      secv_o(LL, 1, -1, 0, 0, S, D).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%cut_o(L:list, I:int, S:int, D:int, C:list, R:list)
%taie lista între indicii S și D

cut_o([H|T], I, S, D, C, R):-
      I >= S,
      D >= I,
      !,
      add_f(C, H, CC),
      IPP is I+1,
      cut_o(T, IPP, S, D, CC, R).


cut_o([], _, _, _, C, C).
cut_o([_|T], I, S, D, C, R):-
      IPP is I + 1,
      cut_o(T, IPP, S, D, C, R).

cut(L, S, D, R):- cut_o(L, 1, S, D, [], R).
```

```prolog
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%afișarea secventei
%run(L:list)
%i determinist

run(L):-
      secv(L, S, D),
      cut(L, S, D, R),
      print(R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```prolog
%sortare cu pastrarea dublurilor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
add_f([], E, [E]).
add_f([H|T], E, [H|R]):-
      add_f(T, E, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%min_o(L:list, M:int, R:int)
%calculează în R minimul dintr-o lista L
%M se modifică pe parcurs
%iio iii determinist

min_o([], M, M).

min_o([H|T], M, R):-
      M > H,
      !,
      min_o(T, H, R).

min_o([_|T], M, R):-
      min_o(T, M, R).

%wrapper min(L:list, M:int)
%minimul M din lista L
min([H|T], M):- min_o([H|T], H, M).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%elimina_o(L:list, E:int, C:list, R:list)
%elimina prima apariție a elementului E din lista L
%C colectoare R rezultat
%determinist iiio iiii

elimina_o([], _, C, C).

elimina_o([H|T], E, C, R):-
      H == E,
      !,
      elimina_o(T, 'n', C, R).

elimina_o([H|T], E, C, R):-
      add_f(C, H, CC),
      elimina_o(T, E, CC, R).

elimina(L, E, R):- elimina_o(L, E, [], R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%sortare_o(L:list, C:list, R:list)
%sortează lista L în R
%C variabilă colectoare
%iii iio determinist

sortare_o([], C, C).

sortare_o(L, C, R):-
      min(L, M),
      elimina(L, M, LL),
      add_f(C, M, CC),
      sortare_o(LL, CC, R).

sortare(L, R):- sortare_o(L, [], R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%sortare cu eliminarea dublurilor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
add_f([], E, [E]).
add_f([H|T], E, [H|R]):-
      add_f(T, E, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%min_o(L:list, M:int, R:int)
%calculează în R minimul dintr-o lista L
%M se modifică pe parcurs
%iio iii determinist

min_o([], M, M).

min_o([H|T], M, R):-
      M > H,
      !,
      min_o(T, H, R).

min_o([_|T], M, R):-
      min_o(T, M, R).

%wrapper min(L:list, M:int)
%minimul M din lista L
min([H|T], M):- min_o([H|T], H, M).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%elimina_o(L:list, E:int, C:list, R:list)
%elimina toate aparițiile elementului E din lista L
%C colectoare R rezultat
%determinist iiio iiii

elimina_o([], _, C, C).

elimina_o([H|T], E, C, R):-
      H == E,
      !,
      elimina_o(T, E, C, R).

elimina_o([H|T], E, C, R):-
      add_f(C, H, CC),
      elimina_o(T, E, CC, R).

elimina(L, E, R):- elimina_o(L, E, [], R).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%sortare_o(L:list, C:list, R:list)
%sortează lista L în R
%C variabilă colectoare
%iii iio determinist

sortare_o([], C, C).

sortare_o(L, C, R):-
      min(L, M),
      elimina(L, M, LL),
      add_f(C, M, CC),
      sortare_o(LL, CC, R).

sortare(L, R):- sortare_o(L, [], R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%să se scrie un predicat care înlocuiește
%un element cu o altă listă data.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%add_f(L:lista, E:int, C:lista)
%adauga la final elementul L
%iii, iio determinist
add_f([], E, [E]).
add_f([H|T], E, [H|C]):-add_f(T, E, C).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%concat(A:lista, B:lista, C:lista, O:lista)
%concatenează A și B
%C colectoare
%O rezultat
%iiio, iiii determinist

concat([], [], C, C).

concat([H|T], B, C, O):-
      add_f(C, H, R),
      concat(T, B, R, O).

concat([], [H|T], C, O):-
      add_f(C, H, R),
      concat([], T, R, O).
```

```
%concatenare full
concatenare(A, B, C):-concat(A, B, [], C).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%sub(A:lista, L:lista, C:lista, O:lista, E:int)
%substituie toate elementele E din A cu lista L
%C colectoare
%O rezultat
%iiioi, iiiii  determinist

sub([], _, C, C, _).

sub([H|T], L, C, O, E):-
      H == E,
      !,
      concatenare(C, L, R),
      sub(T, L, R, O, E).

sub([H|T], L, C, O, E):-
      concatenare(C, [H], R),
      sub(T, L, R, O, E).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%substituie(A:lista, L:lista, E:int, C:list)
%wrapper
substituie(A, L, E, C):-
      sub(A, L, [], C, E).

%pretty print
run(A, L, E):-
      substituie(A, L, E, C),
      write(C).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%determina succesorul nr reprezentat pe lista
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
add_f([], E, [E]).
add_f([H|T], E, [H|R]):-
      add_f(T, E, R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%succ(N:list, C:list, R:list, TR:int, E:int)
%aduna 1 la N
%N numar reprezentat invers pe lista
%colectoare C
%R rezultat
%TR transportul la suma cu 1
%E prima dată e 1, apoi se mută la 0 pe toată execuția.
%iio iii determinist

succ([],C, R, 1, _):-!,
      add_f(C, 1, R).

succ([], C, C, 0, _).

succ([H|T], C, R, TR, E):-
      SUMA is H + E + TR,
      SUMA >= 10,
      !,
      UC is SUMA mod 10,
      add_f(C, UC, CC),
      succ(T, CC, R, 1, 0).

succ([H|T], C, R, TR, E):-
      SUMA is H + E + TR,
      add_f(C, SUMA, CC),
      succ(T, CC, R, 0, 0).

%succesor(L:list, R:list)
%ii io determinist
succesor(L, R):-
      reverse(L, LL),
      succ(LL, [], RR, 0, 1),
      reverse(R, RR).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%suma a două numere reprezentate pe listă
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%reverse_o(L, C, R)

reverse_o([], C, C).
reverse_o([H|T], C, R):-
      reverse_o(T, [H|C], R).

reverse(L, R):- reverse_o(L, [], R).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%suma(A:list, B:list, T:int, S:int, P:int, R:int)
%calculează suma nr A și B reprezentate pe liste
%A și B sunt scrise invers
%T transportul
%S suma care crește
%P puterea
%R rezultatul
%iiiio iiiii determinist

suma([], [], _, S,_, S).

suma([], [H|T], TR, S, P, R):-
      PP is P*10,
      SUMA is ((H + TR)* P) + S,
      suma([], T, 0, SUMA, PP, R).

suma([H|T], [], TR, S, P, R):-
      PP is P*10,
      SUMA is ((H + TR)* P) + S,
      suma(T, [], 0, SUMA, PP, R).


suma([A|TA], [B|TB], TR, S,P, R):-
      PP is P* 10,
      SABT is A + B + TR,
      SABT > 10,
      !,
      SUMA is ((SABT mod 10) * P) + S,
      suma(TA, TB, 1, SUMA, PP, R).

suma([A|TA], [B|TB], TR, S, P, R):-
      PP is P* 10,
      SUMA is ((A + B + TR)* P) + S,
      suma(TA, TB, 0, SUMA, PP, R).
```

```
%wraper run(A:list, B:list, S:int)
%calculează suma
%A și B nr reprezentate pe listă în ordine corectă
run(A, B, S):-
      reverse(A, AR),
      reverse(B, BR),
      suma(AR, BR, 0, 0, 1, S).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
```