```
%--------------------------------------------------------------------------------
%-- Project 1, a Prolog version of the PR-1 program of Gottfried Michael Koenig
%-- interpretation and programming: W.G. Vree, 2007
%--------------------------------------------------------------------------------
%-- For each "musical parameter" this program calculates a sequence of Parts.
%-- The musical parameters are:
%-- chords, duration (called entry-delay) dynamics and tempo.
%-- A Part can be either a Row a Group or a Balance.
%-- Rows and Groups are sequences that obey certain rules of serial music.
%-- A Balance is a (balanced) mixture of Rows and Groups.
%--------------------------------------------------------------------------------

%------------------------------------------------------------------------------
%-- LEVEL 1 (parameter definitions, to be choosen by the composer)------------
%------------------------------------------------------------------------------

notes( ['c', 'c#', 'd', 'd#', 'e', 'f', 'f#', 'g', 'g#', 'a', 'a#', 'b']).

major( [[0, 1, 5], [3, 4, 8], [6, 7, 11], [9, 10, 2]]). % interval-rows for computing chords
minor( [[0, 1, 4], [2, 3, 6], [5, 8, 9], [7, 10, 11]]).

% the possible entry-delays (chord duration) with corresponding minimum and maximum chord size

mapping( [['1/1', 1, 6], ['4/5', 1, 6], ['3/4', 1, 6], ['2/3', 1, 6], ['5/8', 1, 6], ['3/5', 1, 6],
          ['1/2', 1, 4], ['2/5', 1, 4], ['3/8', 1, 4], ['1/3', 1, 4],
          ['1/4', 1, 3], ['1/5', 1, 3], ['1/8', 1, 2], ['0/0', 1, 1],
          ['1/2', 1, 4], ['2/5', 1, 4], ['3/8', 1, 4], ['1/3', 1, 4],
          ['1/4', 1, 3], ['1/5', 1, 3], ['1/8', 1, 2], ['0/0', 1, 1],
          ['1/4', 1, 3], ['1/5', 1, 3], ['1/8', 1, 2], ['0/0', 1, 1],
          ['1/8', 1, 2], ['0/0', 1, 1] ]).

entry_list( X) :- mapping( Y), maplist( fst, Y, X).  % extracted list of possible chord durations

dyna_list( [ppp, pp, p, mp, mf, f, ff, fff]).        % possible dynamics

tempo_list( [60, 52, 45.5, 39.5, 34.5, 30]). % possible tempo values (t60 == 60 1/4-beats per minute)

% the following process-numbers specify the type of serial generation process (see: Level 4)
% 1..3 == Rows are generated
% 4 == Balance structures are generated
% 5..7 == Groups are generated

dyna_process( 4).
entry_process( 4).
chord_process( 4).

rR( 1).             % 1..2, repetition rate for chord rows and groups

% the average number of notes in a chord, caculated from the mapping above

average_chord_size( X) :-  mapping( Ms),
                           length( Ms, Len),
                           maplist( snd, Ms, Mins),
                           maplist( trd, Ms, Maxs),
                           sumlist( Maxs, MaxSum),
                           sumlist( Mins, MinSum),
                           X is (MaxSum + MinSum) / (2 * Len).

%------------------------------------------------------------------------------
%-- LEVEL 1 Section definition ------------------------------------------------
%------------------------------------------------------------------------------

section( N) :- dyna_str( Ds), entry_str( Es), tempo_str( Ts, Es),
               ch_size_str( Sizes, Es),  chord_str( Chords, Sizes),
               output( N, Ds, Es, Ts, Chords).

dyna_str( Parts) :-  dyna_list( Ds),
                     mkRndStr( 11, Rs1),
                     mkRndStr( 12, Rs2),
                     mkRndStr( 13, Rs3),
                     mk_row_str( Ds, Row_str, Rs1),
                     mk_serial_str( Ds, Grp_str, Rs2),
                     dyna_process( Process),
                     length( Ds, Ln),
                     d_section( dynamics, Row_str, Grp_str, Process, Ln, Parts, Rs3).

entry_str( Parts) :- entry_list( Es),
                     mkRndStr( 14, Rs1),
                     mkRndStr( 15, Rs2),
                     mkRndStr( 16, Rs3),
                     mk_row_str( Es, Row_str, Rs1),
                     mk_serial_str( Es, Grp_str, Rs2),
                     entry_process( Process),
                     length( Es, Ln),
                     d_section( entry_delay, Row_str, Grp_str, Process, Ln, Parts, Rs3).

ch_size_str( Parts, Entry_str) :-
                     mkRndStr( 17, Rs1),
                     flat_str( Entry_str, Es),
                     mapping( Ms),
                     d_section( chord_size, Es, [], 0, Ms, Parts, Rs1).
```

```
tempo_str( Parts, Entry_str) :-
                     mkRndStr( 18, [R|Rs1]),
                     flat2( Entry_str, Es),
                     tempo_list( Ts),
                     mk_serial_str( Ts, Grp_str, Rs1),
                     between2( 1, 4, Rep, R),
                     d_section( tempo_grp, Es, Grp_str, 0, Rep, Parts, _).

chord_str( Parts, Ch_size_str) :-
                     mkRndStr( 19, Rs1),
                     mkRndStr( 20, Rs2),
                     notes( Ns),
                     mk_serial_str( Ns, Notes_str, Rs1),
                     chord_process( Process),
                     average_chord_size( A),
                     d_section( chord, Ch_size_str, Notes_str, Process, A, Parts, Rs2).

%-----------------------------------------------------------------------------
%-- LEVEL 2 --General stream creation patterns-------------------------------
%-----------------------------------------------------------------------------

d_section( Par_pred, Row_str, Serial_str, Pn, A, [Part | Parts], Rs1) :-
    call( Par_pred, Row_str, Serial_str, Pn, A, Part, Row_str_rest, Serial_str_rest, Rs1, Rs2),
    freeze( Parts, d_section( Par_pred, Row_str_rest, Serial_str_rest, Pn, A, Parts, Rs2)).

mk_row_str( Par_list, [Row | Row_str], Rs1) :-
    perm( Par_list, Row, Rs1, Rs2),
    freeze( Row_str, mk_row_str( Par_list, Row_str, Rs2)).

mk_serial_str( Par_list, Serial_str, Rs1) :-
    mk_row_str( Par_list, Row_str, Rs1),
    flat_str( Row_str, Serial_str).

%-----------------------------------------------------------------------------
%-- LEVEL 3 --Definition of the parameter functions--------------------------
%-----------------------------------------------------------------------------

dynamics( Row_str, Serial_str, Process, Par_len, Part, Row_str_rest, Serial_str_rest, Rs1, Rs1) :-
    Process =< 3,
    M is round( max( 1, ((Par_len * (5 - Process)) / 4))),
    row( Row_str, Serial_str, M, Part, Row_str_rest, Serial_str_rest).

dynamics( Row_str, Serial_str, Process, Par_len, Part, Row_str_rest, Serial_str_rest, Rs1, Rs2) :-
    Process == 4,
    M is min( 8, Par_len),
    balance( Row_str, Serial_str, 2, M, 1, Part, Row_str_rest, Serial_str_rest, Rs1, Rs2).

dynamics( Row_str, Serial_str, Process,       _, Part, Row_str_rest, Serial_str_rest, [R|Rs1], Rs1) :-
    Process > 4,
    P1 is Process - 4, rR( RR),
    arg( RR, t(t(4, 6,  8), t(4,  6,  8)), MinVec),
    arg( RR, t(t(6, 9, 12), t(10, 15, 20)), MaxVec),
    arg( P1, MinVec, Min),
    arg( P1, MaxVec, Max),
    between2( Min, Max, M, R),
    group( Row_str, Serial_str, M, Part, Row_str_rest, Serial_str_rest).

entry_delay( Row_str, Serial_str, Process, Par_len, Part, Row_str_rest, Serial_str_rest, Rs1, Rs1) :-
    Process =< 3,
    M is round( max( 1, ((Par_len * (5 - Process)) / 4))),
    row( Row_str, Serial_str, M, Part, Row_str_rest, Serial_str_rest).

entry_delay( Row_str, Serial_str, Process, Par_len, Part, Row_str_rest, Serial_str_rest, [R|Rs1], Rs2) :-
    Process == 4,
    M is Par_len // 2,
    between2( 1, 4, Rep, R),
    balance( Row_str, Serial_str, 1, M, Rep, Part, Row_str_rest, Serial_str_rest, Rs1, Rs2).

entry_delay( Row_str, Serial_str, Process,       _, Part, Row_str_rest, Serial_str_rest, [R|Rs1], Rs1) :-
    Process > 4,
    P1 is Process - 4, rR( RR),
    arg( RR, t(t(2, 5,  8), t(3,  7, 11)), MinVec),
    arg( RR, t(t(4, 8, 12), t(6, 11, 16)), MaxVec),
    arg( P1, MinVec, Min),
    arg( P1, MaxVec, Max),
    between2( Min, Max, M, R),
    group( Row_str, Serial_str, M, Part, Row_str_rest, Serial_str_rest).

chord_size( [Delay | Rest_delays], Ys, _, Mapping, Ch_sz, Rest_delays, Ys, [R|Rs1], Rs1) :-
    sublist( fst_eq( Delay), Mapping, Ranges),
    maplist( snd, Ranges, [Min|_]),
    maplist( trd, Ranges, [Max|_]),
    between2( Min, Max, Ch_sz, R).

tempo_grp(      Entries , [_ | Tempos],       _,   0,   [], Entries,        Tempos, Rs1, Rs1) :- !.
tempo_grp( [E | Entries], [T | Tempos], Process, Rtc, Part, Entry_rest, Tempo_rest, Rs1, Rs2) :-
    Rtc1 is Rtc - 1,
    length( E, Len),
    replicate( Len, T, Ts),
    tempo_grp( Entries, [T|Tempos], Process, Rtc1, Ts_rest, Entry_rest, Tempo_rest, Rs1, Rs2),
    append( Ts, Ts_rest, Part).

chord( Row_str, Serial_str, Process, _, Part, Row_str_rest, Serial_str_rest, Rs1, Rs2) :-
    Process =< 3,
    M is 12 - (3 * (Process - 1)),
    row_chord( Row_str, Serial_str, M, Part, Row_str_rest, Serial_str_rest, Rs1, Rs2).

chord( Row_str, Serial_str, Process, Average_ch_size, Part, Row_str_rest, Serial_str_rest, Rs1, Rs2) :-
    Process == 4,
    balance_chord( Row_str, Serial_str, Average_ch_size, Part, Row_str_rest, Serial_str_rest, Rs1, Rs2).

chord( Row_str, Serial_str, Process, Average_ch_size, Part, Row_str_rest, Serial_str_rest, [R1,R2|Rs1], Rs2) :-
    Process >  4,
    rR( RR),
    Min_tone   is Process - 3,
    Max_tone   is Min_tone * (RR + 1),
    Min_group  is round(( 2.0 * Average_ch_size * Min_tone) / 2.25),
    Max_group  is Min_group * (RR + 1),
    Min_dgroup is round((       Average_ch_size * Min_tone) / 2.25),
    Max_dgroup is Min_dgroup * (RR + 1),
    between2( Min_tone,   Max_tone,   Ng1, R1),
    between2( Min_group,  Max_group,  Ng2, R1),
    between2( Min_dgroup, Max_dgroup, Ng3, R1),
    one_of_3( 1, 2, 3, X, R2),
    (X == 1 -> grp_tones( Row_str, Serial_str, Ng1,    Part, Row_str_rest, Serial_str_rest, Rs1, Rs2);
     X == 2 -> grp_chord( Row_str, Serial_str, Ng2, 3, Part, Row_str_rest, Serial_str_rest, Rs1, Rs2);
               grp_chord( Row_str, Serial_str, Ng3, 6, Part, Row_str_rest, Serial_str_rest, Rs1, Rs2)).

%-----------------------------------------------------------------------------
%-- LEVEL 4: Row, Group and Balance of non-chord parameters-------------------
%-----------------------------------------------------------------------------

row(  [Row1 | Row_str], Serial_str, M,                Row, Row_str,  Serial_str ) :-
    take( M, Row1, Row).

group( Row_str, [Ser | Serial_str], M,              Group, Row_str,  Serial_str ) :-
    replicate( M, Ser, Group).

balance( Row_str, Serial_str, Min, Max, Rep, [Set, Bal_p], Row_str2, Serial_str2, Rs1, Rs4) :-
    mk_len_list( Min, Max, Rep, Len_list, Rs1, Rs2),
    mk_proc_list( Rep, Proc_list, Rs2, Rs3),
    maplist( fst, Proc_list, Set_proc_list),
    maplist( snd, Proc_list, Bal_proc_list),
    struc( Set_proc_list, Row_str,  Serial_str,  Len_list, Set, Row_str1, Serial_str1   ),
    struc( Bal_proc_list, Row_str1, Serial_str1, Len_list, Bal, Row_str2, Serial_str2),
    perm( Bal, Bal_p, Rs3, Rs4).

mk_len_list(   _,   _, 0, [], Rs1, Rs1) :- !.
mk_len_list( Min, Max, N, [L | Ls], [R|Rs1], Rs2) :-
    between2( Min, Max, L, R),
    N1 is N - 1,
    mk_len_list( Min, Max, N1, Ls, Rs1, Rs2).

mk_proc_list( 0, [], Rs1, Rs1) :- !.
mk_proc_list( N, [L | Ls], [R|Rs1], Rs2) :-
    one_of( [row, group], [group, row], L, R),
    N1 is N - 1,
    mk_proc_list( N1, Ls, Rs1, Rs2).

struc(      _, Row_str, Serial_str,     [],              [], Row_str,  Serial_str) :- !.
struc( [P|Ps], Row_str, Serial_str, [L|Ls], [Part | Parts], Row_str2, Serial_str2) :-
    call( P, Row_str, Serial_str, L, Part, Row_str1, Serial_str1),
    struc( Ps, Row_str1, Serial_str1, Ls, Parts, Row_str2, Serial_str2).

%-----------------------------------------------------------------------------
%-- LEVEL 4: row, group, double-group, tone and balance of the chord parameter
%-----------------------------------------------------------------------------

row_chord( Ch_len_str, Note_str, Row_size, Chords, Rest_ch_len_str, Rest_note_str, Rs1, Rs2) :-
    trio_row( Note_str, Trio_notes, Rest_note_str, Rs1, Rs2),
    take( Row_size, Trio_notes, Row),
    fill_chord( Ch_len_str, Row, Chords, Rest_ch_len_str).

grp_chord( Ch_len_str, Note_str, Ngroups, Group_size, Chords, Rest_ch_len_str, Rest_note_str, Rs1, Rs2) :-
    trio_row( Note_str, Trio_notes, Rest_note_str, Rs1, Rs2),
    take( Group_size, Trio_notes, Group),
    replicate( Ngroups, Group, Group_rep),
    flatten( Group_rep, Groups),
    fill_chord( Ch_len_str, Groups, Chords, Rest_ch_len_str).

grp_tones( Ch_len_str, [Note | Rest_note_str], Ntones, Chords, Rest_ch_len_str, Rest_note_str, Rs1, Rs2) :-
    mk_tones( Ch_len_str, Note, Ntones, Chords, Rest_ch_len_str, Rs1, Rs2).

mk_tones(           Ch_len_str,     _, 0     , []                       ,      Ch_len_str, Rs1, Rs1) :- !.
mk_tones( [Ch_len | Ch_len_str], Note, Ntones, [Tone_sp | Rest_tones], Rest_ch_len_str, Rs1, Rs3) :-
    trio_row( [Note], Trio_notes, _, Rs1, Rs2),
    take( Ch_len, Trio_notes, [_|Chord]),
    Tone = [Note | Chord],
    concat_atom( Tone, ' ', Tone_sp),
    N1 is Ntones - 1,
    mk_tones( Ch_len_str, Note, N1, Rest_tones, Rest_ch_len_str, Rs2, Rs3).
```

```
balance_chord( Ch_len_str, Note_str, Avrage_ch_size, Chords, Rest_Ch_len_str, Rest_Note_str, [R|Rs1], Rs2) :-
    one_of( 1, 2, X, R),
    bal_chrd( X, Ch_len_str, Note_str, Avrage_ch_size, Chords, Rest_Ch_len_str, Rest_Note_str, Rs1, Rs2).

bal_chrd( 1, Ch_len_str, Note_str, Avrage_ch_size, [[Chords], [Chord_grp]], Ch_len_str_2, Note_str_2, [R1|Rs1], Rs3) :-
    between2( 1, 3, Nrows, R1),
    Ntones is round( (Nrows * 5.0 * 2.25) / Avrage_ch_size),
    Ngroups is round( (Nrows * 4.0 * 2.25) / Avrage_ch_size),
    Ndgroups is round( (Nrows * 2.0 * 2.25) / Avrage_ch_size),
    rep_chord_row( Ch_len_str, Note_str, Nrows, Chords, Ch_len_str_1, Note_str_1, Rs1, Rs2),
    one_of_3( 1, 2, 3, X, R1),
    (X == 1 -> grp_tones( Ch_len_str_1, Note_str_1, Ntones,      Chord_grp, Ch_len_str_2, Note_str_2, Rs2, Rs3);
     X == 2 -> grp_chord( Ch_len_str_1, Note_str_1, Ngroups,  3, Chord_grp, Ch_len_str_2, Note_str_2, Rs2, Rs3);
               grp_chord( Ch_len_str_1, Note_str_1, Ndgroups, 6, Chord_grp, Ch_len_str_2, Note_str_2, Rs2, Rs3)).

bal_chrd( 2, Ch_len_str, Note_str, Avrage_ch_size, [[Chord_grp], [Chords]], Ch_len_str_2, Note_str_2, [R1|Rs1], Rs3) :-
    one_of_3( 6, 4, 2, MinElem, R1),
    rR( RR),
    Mx1 is 18 * RR,
    Mx2 is round( (12.0 * RR * Avrage_ch_size) / 2.25),
    Mx3 is round( (6.0  * RR * Avrage_ch_size) / 2.25),
    one_of_3( Mx1, Mx2, Mx3, MaxElem, R1),
    between2( MinElem, MaxElem, Nelems, R1),
    one_of_3( 5.0, 4.0, 2.0, Xd, R1),
    Nrows is round( Nelems / Xd),
    one_of_3( 1, 2, 3, X, R1),
    (X == 1 -> grp_tones( Ch_len_str, Note_str, Nelems,    Chord_grp, Ch_len_str_1, Note_str_1, Rs1, Rs2);
     X == 2 -> grp_chord( Ch_len_str, Note_str, Nelems, 3, Chord_grp, Ch_len_str_1, Note_str_1, Rs1, Rs2);
               grp_chord( Ch_len_str, Note_str, Nelems, 6, Chord_grp, Ch_len_str_1, Note_str_1, Rs1, Rs2)),
    rep_chord_row( Ch_len_str_1, Note_str_1, Nrows, Chords, Ch_len_str_2, Note_str_2, Rs2, Rs3).

%--------------------------------------------------------------------------
%-- support-functions for the chord parameter functions of level 4 -----------
%--------------------------------------------------------------------------

trio_row( [Start_note | Rest_note_str], Trio_row, Rest_note_str, [R1,R2|Rs1], Rs2) :-
    major( X), minor( Y),
    one_of( X, Y, Interval_row, R1),
    perm( Interval_row, Irp, Rs1, Rs2),
    flatten( Irp, Mixed_row),
    reverse( Mixed_row, Mixed_rev),
    one_of( Mixed_row, Mixed_rev, Trio_list, R2),
    maplist( add_note( Start_note), Trio_list, Trio_row).

fill_chord( [Ch_len | Rest], Row, Chords, Rest_lens) :-
    length( Row, Len),
    Len < Ch_len -> Chords = [],
                    Rest_lens = [Ch_len | Rest]
                 ;
                    append( Chord, Rest_row, Row),
                    length( Chord, Ch_len),
                    fill_chord( Rest, Rest_row, Rest_chords, Rest_lens),
                    concat_atom( Chord, ' ', Chord_sp),
                    Chords = [Chord_sp | Rest_chords].

rep_chord_row( Ch_len_str, Note_str,     0,     [], Ch_len_str,  Note_str , Rs1, Rs1) :- !.
rep_chord_row( Ch_len_str, Note_str, Nrows, Chords, Ch_len_str2, Note_str2, Rs1, Rs3) :-
    N1 is Nrows - 1,
    row_chord(     Ch_len_str,  Note_str,  12, Chords1, Ch_len_str1, Note_str1, Rs1, Rs2),
    rep_chord_row( Ch_len_str1, Note_str1, N1, Chords2, Ch_len_str2, Note_str2, Rs2, Rs3),
    append( Chords1, Chords2, Chords).

%--------------------------------------------------------------------------
%-- SMALL UTILITY FUNCTIONS --------------------------------------------------
%--------------------------------------------------------------------------

one_of(   X1, X2,     Z, R1) :- R is R1 mod 2, nth0( R, [X1, X2    ], Z).

one_of_3( X1, X2, X3, Z, R1) :- R is R1 mod 3, nth0( R, [X1, X2, X3], Z).

between2( Min, Max, M, R) :- M is Min + R mod (1 + Max - Min).

perm( [X], [X],         Rs, Rs ) :- !.
perm(  Xs, [V|Zs], [R|Rs1], Rs2) :-
    length( Xs, Len),
    Len2 is R mod Len,
    append( Us, [V|Vs], Xs),
    length( Us, Len2),
    append( Us, Vs, Ys),
    perm( Ys, Zs, Rs1, Rs2).

mkRndStr( V, [R|Rs]) :- R is random( 65536), freeze( Rs, mkRndStr( V, Rs)).

add_note( Start_Note, Interval, Note) :-
    notes( Notes),
    nth0( Ix, Notes, Start_Note),
    Iy is (Ix + Interval) mod 12,
    nth0( Iy, Notes, Note).

fst( [X|_], X).
```

```
snd( [_,X|_], X).
trd( [_,_,X|_], X).
fst_eq( X, [X|_]).

replicate( 0, _, []) :- !.
replicate( N, E, [E|Es]) :- N1 is N - 1, replicate( N1, E, Es).

flat_str( [Xs|Xss], Zs) :-
    flatten( Xs, Us),
    append( Us, Ys, Zs),
    freeze( Ys, flat_str( Xss, Ys)).

flat2( [[Set,Bal] | Xss], Zs) :- !,
    compound( Set),
    append( Set, Bal, Sbs),
    append( Sbs, Ys, Zs),
    freeze( Ys, flat2( Xss, Ys)).
flat2( Rows, Rows).

take(0, _, []) :- !.
take(N, [X|Xs], [X|Zs]) :- N1 is N-1, take(N1, Xs, Zs).

cmp_list( [X|_]) :- is_list( X).

elems( [X],    N) :- !,
    cmp_list( X) -> plist( X, N)
                 ;  write( X).
elems( [X|Xs], N) :-
    cmp_list( X) -> plist( X, N), nl, tab(N), elems( Xs, N)
                 ;  write( X),    nl, tab(N), elems( Xs, N).

plist( X, N) :- write( '['), N1 is N + 1, elems( X, N1), write( ']').

output( N, Dss, Ess, Tss, Css) :-
    tagStr( 5, ' ', Dss, Tds), tagStr( 5, ' ', Ess, Tes), tagStr( 7, ' ', Tss, Tts),
    tagStr( 5, ' ', Css, Tcs),
    pr_str( N, Tds, Tes, Tts, Tcs).

tagStr( N, B, [[X|Xs] | Xss], Tags) :-
    compound( X) ->
        Set = X, [Bal|_] = Xs,
        mkSbTags( N, 'S', Set, STags),
        mkSbTags( N, 'B', Bal, BTags),
        append( STags, BTags, SbTags),
        append( SbTags, Tags_rest, Tags),
        freeze( Tags_rest, tagStr( N, B, Xss, Tags_rest))
    ;
        mkTags( N, B, [X|Xs], RgTags),
        append( RgTags, Tags_rest, Tags),
        freeze( Tags_rest, tagStr( N, B, Xss, Tags_rest)).

mkSbTags( _, _,         [],   []) :- !.
mkSbTags( N, B, [Xs | Xss], Tags) :-
    mkTags( N, B, Xs, RgTags),
    append( RgTags, Tags_rest, Tags),
    mkSbTags( N, ' ', Xss, Tags_rest).

mkTags( N, B, [X | Xs], [t(N,   B, '*', X) | Rest]) :- mkRest( N, Xs, Rest).
mkRest( _, [], []) :- !.
mkRest( N, [X | Xs],    [t(N, ' ', ' ', X) | Rest]) :- mkRest( N, Xs, Rest).

pr_str( 0, _, _, _, _) :- !.
pr_str( N, [D|Ds], [E|Es], [T|Ts], [C|Cs]) :-
    prt(D, [], As1), prt(E, As1, As2), prt(T, As2, As3), prt(C, As3, As),
    format( '~w~w~w~t~10||~w~w~w~t~20||~w~w~w~t~30||~w~w~w~t~50||~n', As),
    N1 is N - 1,
    pr_str( N1, Ds, Es, Ts, Cs).

prt( t(_, B, Rg, X), Args_in, Args_out) :- append( Args_in, [B, Rg, X], Args_out).
```