```
/*
Prolog code for a naive binary search tree data structure
devoid of any balancing methods (e.g., red-black marking)
and database/ state management (i.e., assert/ retract).

Michael E Sparks, 6 Nov 2020

Let 'tree' ~ functor to construct the data structure,
tree(Tl,R,Tr), where Tl ~ left subtree, R ~ root node,
and Tr ~ right subtree.
Tl and Tr can be empty, in which case they assume the
value 'null'.

SAMPLE USAGE:

?- grow_tree([5,1+2,66,12+3*4,522^3,0,4*(-20),-4],null,T),
|    del_node(9/3,T,T1),
|    trim_tree([5],T1,T2),
|    ins_node(-10,T2,T3),
|    display(T), nl, nl, display_eval(T3),
|    node(3,T3).
          522^3
      66
          12+3*4
  5
      1+2
          0
                  -4
              4* -20


          142236648
      66
          24
  0
          -4
              -10
      -80
false.

?- grow_tree([1,2,3,4,5,6,7,8,9,10],null,T),
|    trim_tree([2,3,1],T,T1),
|    node(10,T1),
|    display(T1).
                          10
                      9
                  8
              7
          6
      5
  4
T = tree(null, 1, tree(null, 2, tree(null, 3, tree(null, 4, tree(null, 5, tree(n
ull, 6, tree(null, 7, tree(null, 8, tree(null, 9, tree(..., ..., ...)))))))))),
T1 = tree(null, 4, tree(null, 5, tree(null, 6, tree(null, 7, tree(null, 8, tree(
null, 9, tree(null, 10, null)))))))).

?- grow_tree([1,2,3,4,5,6,7,8,9,10],null,T),
|    trim_tree([2,3,1],T,T1),
|    node(11,T1),
```

```
|    display(T1).
false.

?- member(10,[4,5,6,7,8,9,10]).
true.

?- member(11,[4,5,6,7,8,9,10]).
false.
*/

% instantiate a new tree w/ root value, N
ins_node(N,null,tree(null,N,null)) :- !.

% New value, N, equals existing root, so 'pass'.
% Can't just use tree(Tl,N,Tr) as arg1 b/c there's
% no guarantee N and/or R aren't expressed as formulas.
% That is, 1 * 3 = 2 + 1 returns false in Prolog.
ins_node(N,tree(Tl,R,Tr),tree(Tl,R,Tr)) :-
    N =:= R, !.

% New value, N, belongs in left subtree
ins_node(N,tree(Tl,R,Tr),tree(Tl1,R,Tr)) :-
    N < R, !,
    ins_node(N,Tl,Tl1).

% New value, N, belongs in right subtree
ins_node(N,tree(Tl,R,Tr),tree(Tl,R,Tr1)) :-
    N > R, !,
    ins_node(N,Tr,Tr1).

% tree of only one node, so just return null
% Technically, this would be subsumed by either of the two
% following clauses (we don't necessarily know Tl (or Tr)
% isn't also null), but here we make this single-node
% tree case plainly explict.
% As mentioned elsewhere, we check using N =:= R b/c
% 1 + 3 = 4 is false, whereas 1 + 3 =:= 4 is true.
del_node(N,tree(null,R,null),null) :-
    N =:= R, !.

% right subtree DNE, so just return left subtree.
del_node(N,tree(Tl,R,null),Tl) :-
    N =:= R, !.

% left subtree DNE, so just return right subtree.
del_node(N,tree(null,R,Tr),Tr) :-
    N =:= R, !.

% remove root and replace it w/ max of left subtree
del_node(N,tree(Tl,R,Tr),tree(Tl1,TlMax,Tr)) :-
    N =:= R, !,
    del_rightmost_node(Tl,TlMax,Tl1).

% remove node from left subtree
del_node(N,tree(Tl,R,Tr),tree(Tl1,R,Tr)) :-
    N < R, !,
    del_node(N,Tl,Tl1).

% remove node from right subtree
```

```
del_node(N,tree(Tl,R,Tr),tree(Tl,R,Tr1)) :-
    N > R, !,
    del_node(N,Tr,Tr1).

% clip off max-valued leaf from tree
% and return the resulting fragments.
del_rightmost_node(tree(Tl,Max,null),Max,Tl) :- !.

del_rightmost_node(tree(Tl,R,Tr),Max,tree(Tl,R,Tr1)) :-
    del_rightmost_node(Tr,Max,Tr1).

% build binary search tree from list of values
grow_tree([],T,T).

grow_tree([Head|Tail],Old,New) :-
    ins_node(Head,Old,Tmp),
    grow_tree(Tail,Tmp,New).

% purge binary search tree of values in list
trim_tree([],T,T).

trim_tree([Head|Tail],Old,New) :-
    del_node(Head,Old,Tmp),
    trim_tree(Tail,Tmp,New).

% pretty print the tree structure
display(Tree) :-
    display(Tree,2).

display(null,_).

display(tree(Tl,R,Tr),D) :-
    D1 is D + 4,
    display(Tr,D1),
    tab(D),
    write(R),
    nl,
    display(Tl,D1).

% as above, but evaluating any formulas
% stored in nodes before printing
display_eval(Tree) :-
    display_eval(Tree,2).

display_eval(null,_).

display_eval(tree(Tl,R,Tr),D) :-
    D1 is D + 4,
    display_eval(Tr,D1),
    tab(D),
    R1 is R,
    write(R1),
    nl,
    display_eval(Tl,D1).

% N is root of search tree
node(N,tree(_,R,_)) :-
    N =:= R, !.
```

```
% N is in left subtree (if at all)
node(N,tree(Tl,R,_)) :-
    N < R, !,
    node(N,Tl).

% N is in right subtree (if at all)
node(N,tree(_,R,Tr)) :-
    N > R, !,
    node(N,Tr).
```