## Learning Abstract:

## Task 1:

Contemplate the nature of the problem, see specification on web page for details.

## Task 2:

Copy and paste source code and check to ensure validity and that it initially compiles, see specification on web page for details, full code posted later this document.

## Task 3: One Move Predicate and a Unit Test

m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-

    Tower1Before = [H|T],

    Tower1After = T,

    Tower2Before = L,

    Tower2After = [H|L].

test\_\_m12 :-

    write('Testing: move\_m12\n'),

    TowersBefore = [[t,s,m,l,h],[],[]],

    trace('','TowersBefore',TowersBefore),

    m12(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

Text

Description automatically generated

## Task 4: The Remaining Five Move Predicates and a Unit Tests

m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-

    Tower1Before = [H|T],

    Tower1After = T,

    Tower2Before = L,

    Tower2After = [H|L].

m13([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-

    Tower1Before = [H|T],

    Tower1After = T,

    Tower3Before = L,

    Tower3After = [H|L].

m21([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-

    Tower2Before = [H|T],

    Tower2After = T,

    Tower1Before = L,

    Tower1After = [H|L].

m23([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-

    Tower2Before = [H|T],

    Tower2After = T,

    Tower3Before = L,

    Tower3After = [H|L].

m31([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-

    Tower3Before = [H|T],

    Tower3After = T,

    Tower1Before = L,

    Tower1After = [H|L].

m32([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-

    Tower3Before = [H|T],

    Tower3After = T,

    Tower2Before = L,

    Tower2After = [H|L].

% --- Unit test programs

test\_\_m12 :-

    write('Testing: move\_m12\n'),

    TowersBefore = [[t,s,m,l,h],[],[]],

    trace('','TowersBefore',TowersBefore),

    m12(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m13 :-

    write('Testing: move\_m13\n'),

    TowersBefore = [[t,s,m,l,h],[],[]],

    trace('','TowersBefore',TowersBefore),

    m13(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m21 :-

    write('Testing: move\_m21\n'),

    TowersBefore = [[],[t,s,m,l,h],[]],

    trace('','TowersBefore',TowersBefore),

    m21(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m23 :-

    write('Testing: move\_m23\n'),

    TowersBefore = [[],[t,s,m,l,h],[]],

    trace('','TowersBefore',TowersBefore),

    m23(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m31 :-

    write('Testing: move\_m31\n'),

    TowersBefore = [[],[],[t,s,m,l,h]],

    trace('','TowersBefore',TowersBefore),

    m31(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m32 :-

    write('Testing: move\_m32\n'),

    TowersBefore = [[],[],[t,s,m,l,h]],

    trace('','TowersBefore',TowersBefore),

    m32(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

A screenshot of a computer

Description automatically generated with medium confidence

## Task 5: Valid State Predicate and Unit Test

% -----------------------------------------------------------------------

% --- valid\_state(S) :: S is a valid state

valid\_state(S) :-

    S = [[t,s,m,l,h],[],[]].

valid\_state(S) :-

    S = [[],[t,s,m,h],[l]].

valid\_state(S) :-

    S = [[],[h],[t,s,m,l]].

%% Unit Test Code

test\_\_valid\_state :-

    write('Testing: valid\_state\n'),

    test\_\_vs([[l,t,s,m,h],[],[]]),

    test\_\_vs([[t,s,m,l,h],[],[]]),

    test\_\_vs([[],[h,t,s,m],[l]]),

    test\_\_vs([[],[t,s,m,h],[l]]),

    test\_\_vs([[],[h],[l,m,s,t]]),

    test\_\_vs([[],[h],[t,s,m,l]]).

test\_\_vs(S) :-

    valid\_state(S),

    write(S), write(' is valid.'), nl.

test\_\_vs(S) :-

    write(S), write(' is invalid.'), nl.

Text

Description automatically generated

## Task 6: Defining the write sequence predicate

%% Write Sequence Doe

write\_sequence([]).

write\_sequence([H|T]) :-

    elaborate(H,E),

    write(E),nl,

    write\_sequence(T).

elaborate(m12,Output) :-

    Output = 'Transfer a disk from tower 1 to tower 2.'.

elaborate(m13,Output) :-

    Output = 'Transfer a disk from tower 1 to tower 3.'.

elaborate(m21,Output) :-

    Output = 'Transfer a disk from tower 2 to tower 1.'.

elaborate(m23,Output) :-

    Output = 'Transfer a disk from tower 2 to tower 3.'.

elaborate(m31,Output) :-

    Output = 'Transfer a disk from tower 3 to tower 1.'.

elaborate(m32,Output) :-

    Output = 'Transfer a disk from tower 3 to tower 2.'.

%% Unit Test Code

test\_\_write\_sequence :-

    write('First test of write\_sequence ...'), nl,

    write\_sequence([m31,m12,m13,m21]),

    write('Second test of write\_sequence ...'), nl,

    write\_sequence([m13,m12,m32,m13,m21,m23,m13]).

Text

Description automatically generated