COMP90048 proj2 mppan **LOG** Page 1/1

```
Num Test                                Secs Status    Score    Remark
--- ----                                ---- ------    -----    ------
  1 puzzle_solution(inout) ...          0.01   PASS   1.0/1.0
  2 puzzle_solution(inout) ...          0.01   PASS   1.0/1.0
  3 puzzle_solution(inout) ...          0.04   PASS   1.0/1.0
  4 puzzle_solution(inout) ...          0.06   PASS   1.0/1.0
  5 puzzle_solution(inout) ...          0.04   PASS   1.0/1.0
  6 puzzle_solution(inout) ...          0.07   PASS   1.0/1.0
  7 puzzle_solution(inout) ...          0.05   PASS   1.0/1.0

Total tests executed: 7
Total correctness : 7.00 / 7.00 = 100.00%
Marks earned : 10.50 / 10.50
```

```
% File           : proj2.pl
% Author         : Ming Pan
% Start Date : 10/05/2019
% Purpose        : COMP90048 Declarative Programming Project 2 submission

% library used for this project
:- ensure_loaded(library(clpfd)).

% Authorship Decleration:
% This code document is used for solving the "maths puzzle" problem in
% the subject with code COMP90048 provided by the University of Melbourne.
% The code is written in prolog by
% Ming Pan, Student of University of Melbourne (Master of IT).
% Copyright: Â© 2019 Ming Pan, all rights reserved.

% Project explaination & solution strategy:
% A maths puzzle is a square grid of squares, each to be filled in with a
% single digit from 1 to 9 and satisfied the following constraints:
%   1. each row and each column contains no repeated digits.
%   2. all squares on the diagonal line from upperleft to lower right
%      contain the same value.
%   3. the heading of each row and column holds either the sum or the
%      product of all the digits in that row or column.
% ------------------------------------------------------------------------
% For example, a given puzzle:
%        | 14 | 10 | 35 |
%    14  |    |    |    |
%    15  |    |    |    |
%    28  |    | 1  |    |
% ------------------------------------------------------------------------
% should have a solution as:
%        | 14 | 10 | 35 |
%    14  | 7  | 2  | 1  |
%    15  | 3  | 7  | 5  |
%    28  | 4  | 1  | 7  |
% ------------------------------------------------------------------------
% NOTE: only ONE solution needed for each puzzle.
% ------------------------------------------------------------------------
% The solution proposed in this document is to find all the contraints and
% apply them to the puzzle_solution/1 predicate and use maplist/2 and
% label/1 to obtain a ground solution for that puzzle.

% this predicate check if the sum/product of the elements in a list is
% equal to it's first elements, for example:
% 1. A list: [14, 2, 7, 1] should be true since the header value is the
%    product of all other elements in the list.
% 2. A list: [14, 3, 6, 5] should be true since the header value is the
%    Sum of all other elements in the list.
% 3. A list: [14, 3, 5, 6] should be false since the header value is not
%    the sum nor the product of other lements
check_result([]).
check_result([X|Xs]):-
        check_sum(Xs, X);
        check_product(Xs, X).

% check if the sum of the list elements is equal to the Sum given
check_sum([],0).
check_sum([X|Xs], Sum):-
        check_sum(Xs, TailSum),
```

```
        Sum #= X + TailSum.

% check if the product of the list elements is equal to the Product given
check_product([],1).
check_product([X|Xs], Product):-
        check_product(Xs, TailProduct),
        Product #= X * TailProduct.

% the predicate check if the diagonal line from upper left to lower
% right contain the same value. For example:
% Row 1:    1    2    3
% Row 2:    4    1    5
% Row 3:    6    7    1
% the structure above should return true since the 3 rows have the
% same value 1 which lined at the diagonal form.
check_diagonal_value([]).
check_diagonal_value([R|Rs]):-
        nth0(1,R,Val),
        check_diagonal_value(Rs, 2, Val).

% use recursive way to check the diagnonal value consistency by passing
% the index and previous value gotten from last row.
check_diagonal_value([], _, _).
check_diagonal_value([R|Rs], Index, Val):-
        nth0(Index, R, Elem),
        NewIndex #= Index + 1,
        Val #= Elem,
        check_diagonal_value(Rs, NewIndex, Val).

% check if all the rows satisfied the constraints recursively.
check_rows_constraints([]).
check_rows_constraints([R|Rs]):-
        check_row_constraints(R),
        check_rows_constraints(Rs).

% check one row if it satisfy all the constraints that is:
% 1. all the tail elements is unique and
% 2. all the tail elements should ranged 1 - 9 and
% 3. the sum/product of the rest elements is equal to the header
check_row_constraints(Row):-
        Row = [_|T],
        all_distinct(T),
        T ins 1..9,
        check_result(Row).

% check column constraints by transfer column to row and test with
% check_rows_constraints predicate.
check_columns_constraints(Puzzle):-
        transpose(Puzzle, [_|Columns]),
        check_rows_constraints(Columns).

% find solution by applying the all the constraints.
% 1. check_diagonal_value take the rows with header of the puzzle excl.
% 2. check_rows_constraints take the rows with the header of the puzzle
%    excl.
% 3. check_columns_constraints take the puzzle since it change all the
%    rows of a puzzle to columns form.
% 4. use maplist/2 and label/1 to make sure that all the variables in the
%    puzzle are grounded.
```

```
puzzle_solution(Puzzle):-
        Puzzle = [_| Rows],
        check_diagonal_value(Rows),
        check_rows_constraints(Rows),
        check_columns_constraints(Puzzle),
        maplist(label, Puzzle).
```