ASSIGNMENT-2

1. ***N Queens Problem***

```
:- use_module(library(clpfd)).

% check for placing the first queen on the board.

chck_if_queens_safe([]).
chck_if_queens_safe([Q|Remaining]) :-
        chck_if_queens_safe(Remaining, Q, 1),
        chck_if_queens_safe(Remaining).

% diagonal check for queens safety
% D will check if queen is safe along the diagonal
% Q+D --> up through the diagonal from Q
% Q-D --> down through the diagonal from Q

chck_if_queens_safe([], _, _).
chck_if_queens_safe([Q|Remaining], Q1, D) :-
        Q1 #\= Q+D,
           Q1 #\= Q-D,
        D1 #= D + 1,
        chck_if_queens_safe(Remaining, Q1, D1).

queens(N, Qs) :-
        length(Qs, N),              % length of the list Qs (depends on no of queens)
        Qs ins 1..N,                % domian of Qs
        all_distinct(Qs),  % this will place queens in different rows and columns
```

```
    labeling([ff],Qs),        % labelling in first fail fashion.
    chck_if_queens_safe(Qs).
```

SAMPLE RUN:

```
? queens(8, Qs).
Qs = [1, 5, 8, 6, 3, 7, 2, 4]
0.878 seconds cpu time
Qs = [1, 6, 8, 3, 7, 4, 2, 5]
0.261 seconds cpu time
Qs = [1, 7, 4, 6, 8, 2, 5, 3]
0.170 seconds cpu time
Qs = [1, 7, 5, 8, 2, 4, 6, 3]
```

2. *Sudoku Solver*

```
:- use_module(library(clpfd)).

sudoku(Rows):-
    Rows = [R1,R2,R3,R4,R5,R6,R7,R8,R9],
    problem(1, Rows),
    display_listoflist(Rows,Elements),

    Elements ins 1..9,
    set_rows(Rows),
    set_Cols(R1,R2,R3,R4,R5,R6,R7,R8,R9),
    set_blocks(R1,R2,R3),
```

```
        set_blocks(R4,R5,R6),
        set_blocks(R7,R8,R9),
        (labeling([ff],Elements) -> true ).      % labelling in first fail fashion

%The rows of the sudoku are set according to the constraints

set_rows([]).
set_rows([R1|Rest]):-
        all_distinct(R1),         % each row had disticnt numbers from 1 to 9
        set_rows(Rest).

%The columns of the sudoku are set according to the constraints

set_Cols([],[],[],[],[],[],[],[],[]).
set_Cols([N1|R1], [N2|R2], [N3|R3], [N4|R4], [N5|R5], [N6|R6],
[N7|R7], [N8|R8], [N9|R9]):-
   % each column has distinct nos from 1 to 9
        all_distinct([N1,N2,N3,N4,N5,N6,N7,N8,N9]),
   set_Cols(R1,R2,R3,R4,R5,R6,R7,R8,R9).

% To set blocks of the sudoku and put constraints on each block recursively

set_blocks([],[],[]).
set_blocks([N1,N2,N3|R1], [N4,N5,N6|R2], [N7,N8,N9|R3]):-
   %each block has distinct numbers from 1 to 9
   all_distinct([N1,N2,N3,N4,N5,N6,N7,N8,N9]),
        set_blocks(R1,R2,R3).
```

```
% To display sudoku solver for the given problem in a particular fashion
% by appending and recursively.

display_listoflist([],[]).          % base case
display_listoflist([C|Rest],B):-
        append(C,A,B),          % combines 2 list to form a single list
        display_listoflist(Rest,A).


% this is the given sudoku problem

problem(1, [[_,_,6, 5,9,_, _,_,_],
                        [_,_,3, _,_,_, _,7,_],
                        [_,_,_, _,_,_, 5,6,_],
                        [_,2,_, 1,7,_, _,_,_],
                        [4,8,5, _,_,_, _,_,_],
                        [_,6,_, _,_,4, 9,_,_],
                        [2,_,_, _,_,5, _,_,8],
                        [_,3,8, _,_,1, _,_,_],
                        [_,_,_, 3,_,_, 7,5,4]]).
```

SAMPLE RUN:

***Rows** = [[8, 1, 6, 5, 9, 7, 4, 3, 2], [9, 5, 3, 4, 2, 6, 8, 7, 1], [7, 4, 2, 8, 1, 3, 5, 6, 9], [3, 2, 9, 1, 7, 8, 6, 4 , 5], [4, 8, 5, 6, 3, 9, 1, 2, 7], [1, 6, 7, 2, 5, 4, 9, 8, 3], [2, 7, 4, 9, 6, 5, 3, 1, 8], [5, 3, 8, 7, 4, 1, 2, 9, 6], [6, 9, 1, 3, 8, 2, 7, 5, 4]]*

3. Zebra Puzzle

```
:-use_module(library(clpfd)).

solveZebra(Water,Zebra):-

   % all 25 Components: 5-colours; 5-nationalities; 5-drinks; 5-pets; 5-cig brands

Components=[Red, Green, White, Yellow, Blue, English, Spanish, Dog,
Coffee, Ukrainian, Tea, WinstonSmoker, Serpent, KoolSmoker,
Milk, Norwegian, ChesterfieldSmoker, Fox, Horse, LuckystrikeSmoker,
Juice, Japanese, KentSmoker, Zebra, Water],

   % all given constrains are mentioned in clpfd fashion

   %There are five colored houses in a row (numbered 1 to 5), each with an owner, a pet,
     %cigarettes, and a drink.

     Components ins 1..5,

   % The English lives in the red house.
     English#=Red,

   %The Spanish has a dog.
     Spanish #= Dog,

   %They drink coffee in the green house.
     Coffee #= Green,
```

%The Ukrainian drinks tea.

Ukrainian #= Tea,

%The green house is next to the white house.Can be to right or left

Green #= White+1 #\/ Green #= White-1,

% The Winston smoker has a serpent.

WinstonSmoker #= Serpent,

% In the yellow house they smoke Kool.

Yellow #= KoolSmoker,

% In the middle house they drink milk.Out of 5, 3 is the middle number

Milk #= 3,

%The Norwegian lives in the first house from the left.

Norwegian #= 1,

%The Chesterfield smoker lives near the man with the fox

ChesterfieldSmoker #= Fox+1 #\/ ChesterfieldSmoker #= Fox-1,

%. In the house near the house with the horse they smoke Kool.

KoolSmoker #= Horse+1 #\/ KoolSmoker #= Horse-1,

% The Lucky Strike smoker drinks juice

LuckystrikeSmoker #= Juice,

%The Japanese smokes Kent.

```
    Japanese #= KentSmoker,

    %The Norwegian lives near the blue house.
    Norwegian #= Blue+1 #\/ Norwegian #= Blue-1,

    % all these below mentioned must be placed in different houses

    % each house has different colour
all_distinct([Red,Green,Yellow,Blue,White]),

    % owner--people with different nationality stays in separate houses
all_distinct([English,Spanish,Ukrainian,Japanese,Norwegian]),

    %each owners owns a distinct pet
all_distinct([Dog,Fox,Horse,Serpent,Zebra]),

    % each owner drinks different drinks
all_distinct([Coffee,Water,Milk,Juice,Tea]),

    %each owner in the house smokes different brands of ciggerett
all_distinct([ChesterfieldSmoker,LuckystrikeSmoker,KentSmoker,KoolSmoker,WinstonSmo
ker]),

label(Components).
```

SAMPLE RUN:

solveZebra(Water,Zebra).

***Water** = 1,*
***Zebra** = 4*

***Water** = 1,*
***Zebra** = 5*

4. MAP COLOURING

```
:- use_module(library(clpfd)).

% Four different colours required to colour the map.
% each colour is associated with a number which is easy to map.

color(red,1).
color(green,2).
color(blue,3).
color(yellow,4).

color_map(L):-

        % it has 6 different regions

        A=[A1,A2,A3,A4,A5,A6],
        A ins 1..4,       % 4 different colours -- domain

        % this gives the list of all adjacent sides so they shuld be
        % coloured with different colours

    all_different([A1,A2]),
    all_different([A1,A3]),
    all_different([A1,A4]),
       all_different([A1,A6]),
    all_different([A2,A3]),
    all_different([A2,A5]),
    all_different([A3,A4]),
    all_different([A3,A5]),
       all_different([A3,A6]),
    all_different([A4,A5]),
    all_different([A4,A6]),
```

```
        % to display list in the required fashion
        display_final_list(L),
        region_map(A,L).

    % To display final list as list_of_list

    display_final_list(A):-A=[[1,_],[2,_],[3,_],[4,_],[5,_],[6,_]].

    % recursive function to assign colours to the region which is obtained
    % after checking all constraints.

    region_map([], []).
    region_map([A1|R],[[_,X]|R1]) :- color(X,A1),
                                     region_map(R,R1).
```

SAMPLE RUN:

```
color_map(L).

L = [[1, red], [2, green], [3, blue], [4, green], [5, red], [6, yellow]]
L = [[1, red], [2, green], [3, blue], [4, green], [5, yellow], [6, yellow]]
L = [[1, red], [2, green], [3, blue], [4, yellow], [5, red], [6, green]]
L = [[1, red], [2, green], [3, yellow], [4, green], [5, red], [6, blue]]
```