Assignment-4 Part-A: Constraint Logic Programming

Total Points = 40

Instructions:

1. You must submit this assignment on Canvas by clicking the title link for the assignment. This is a **Constraint Logic Programming over Finite Domain - CLP(FD)** assignment to be programmed using **clp(fd) library in SWI Prolog**. Make sure your programs compile (without any compiler errors). You will not receive credit if your program for a problem does not compile. If you are unable to complete any of the programs, submit the parts that work (with no compiler errors) for partial credit.
2. Your grade will be based on meeting the requirements, functionality (does the program do what it is supposed to do), readability (is the code nicely indented and formatted), and understandability (are the literals meaningful and is the code well documented with appropriate comments). You have to incorporate all the good programming practices and styles.
3. **You are not allowed to use higher order predicates like maplist, findall, aggregation predicates, etc. for this assignment.**

Deliverables:

1. You should **submit one .pl file** named **assign4A.pl**. This file should contain your code for the individual problems(s) in the assignment separated by comments for the grader to run. This file will only contain the code and not the sample runs.
2. You should also submit **one PDF file** named **assign4A.pdf**. This PDF file should contain code to the solution of the problems (in **text format only, no snapshot** will be accepted) and your program's sample runs with output (in **text format only, no snapshot** will be accepted). Make sure to paste your code correctly (with **correct indentation**). If you have any text for the grader to read before grading, you can include it at the top of this file as comments or in canvas notes.

1. N Queens Problem: (5 Points)

Write a CLP prolog program using clp(fd) for the n-queens problem. The objective is to place n queens on a chessboard of size (n* n) so that no two queens are attacking each other; i.e., no two queens are in the same row, the same column, or on the same diagonal.

Note: Represent the positions of the queens as a list of numbers 1…N. For example: When N = 8, the list (Qs) with the positions would be [4,2,7,3,6,8,5,1]. This means that the queen in the first column is in row 4, the queen in the second column is in row 2, etc.
The goal predicate will take the form:
queens (N, Qs).
where N = the number of queens
Qs = solution to the problem

Sample Run:

```
?- queens(8, Qs).
Qs = [4, 2, 7, 3, 6, 8, 5, 1];
Qs = [5, 2, 4, 7, 3, 8, 6, 1];
Qs = [3, 5, 2, 8, 6, 4, 7, 1];
…
…
…
```

2. Sudoku Solver: **(5 Points)**

Write a CLP prolog program using clp(fd) to create a Sudoku solver. Write a predicate named 'sudoku' that takes in an instance of the sudoku problem as a list of 9 rows. Each row is a list of 9 elements (with values from 1 to 9 or an underscore to represent the missing value).

This 'sudoku(Rows)' predicate must be tested on the following Sudoku instance (by including the 'problem' predicate given below in your program). This predicate must then print the complete solved Sudoku. Your code must work even if the Sudoku instance given in the 'problem' predicate is changed in your program.

```
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:

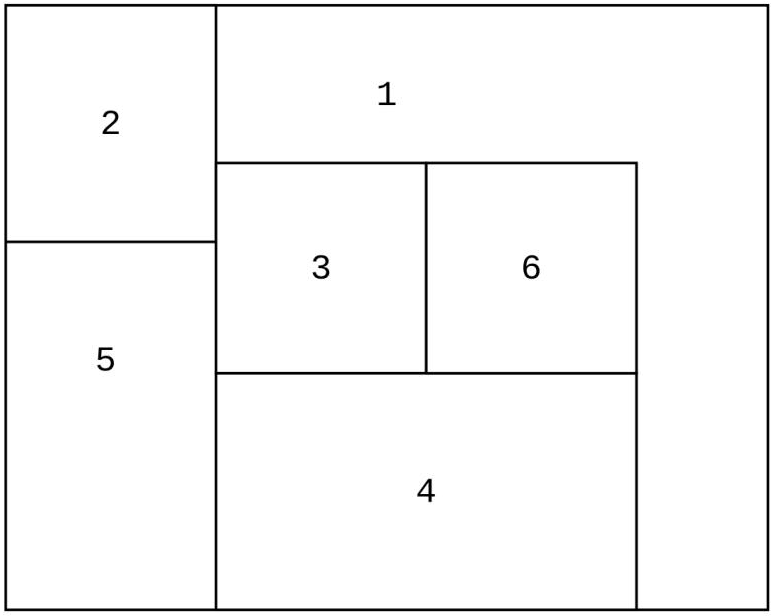
```
?- problem(1, Rows), sudoku(Rows).

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. Map Coloring: **(15 Points)**

A famous problem in mathematics concerns coloring adjacent planar regions. Like cartographic maps, in this problem, it is required that, whatever colors are actually used, no two adjacent regions may have the same color given the total number of colors to be used in the entire region are less than or equal to 4. Two regions are considered adjacent if they share some boundary line segment. Write a prolog program to represent the following region (map) below and write a CLP prolog program "color_map" using clp(fd) that will color the map for the whole region so that no adjacent regions are colored with the same color given that you should use at most 4 colors (red, green, blue, yellow) for the entire region. Any property of the graph instance should not be hardcoded in the "color_map" predicate.

Please note that you need to represent the following region as well in your solution by writing the vertex, edge, and color predicates, for full credit.



Sample Run:

?-color_map(L).
L = [[1, red], [2, green], [3, blue], [4, yellow], [5, red], [6, green]];
L = [[1, red], [2, green], [3, blue], [4, green], [5, yellow], [6, yellow]];
…

4. Zebra Puzzle: **(15 Points)**

Write a CLP prolog program using clp(fd) to solve the following Zebra puzzle and answer which house (number) owns the zebra and in which house (number) the owner drinks water, based on the constraints given in the puzzle:

1. There are five colored houses in a row (numbered 1 to 5), each with an owner, a pet, cigarettes, and a drink.
2. The English lives in the red house.
3. The Spanish has a dog.
4. They drink coffee in the green house.
5. The Ukrainian drinks tea.
6. The green house is next to the white house.
7. The Winston smoker has a serpent.
8. In the yellow house they smoke Kool.
9. In the middle house they drink milk.
10. The Norwegian lives in the first house from the left.
11. The Chesterfield smoker lives near the man with the fox.
12. In the house near the house with the horse they smoke Kool.
13. The Lucky Strike smoker drinks juice.
14. The Japanese smokes Kent.
15. The Norwegian lives near the blue house.

Sample Run:

```
?       - solveZebra(Zebra,Water).
Water = 1
Zebra = 5;

Water = 1
Zebra = 4
```