CS 6374.001 Midterm Exam

There are 5 problems in this exam, each worth 20 points. You have to program **all 4 in Prolog** (either SWI or SICStus Prolog). Please submit your answers in a MS Word file. For each problem, please include:

1. A brief explanation of your solution strategy and your code.
2. The actual Prolog code (along with comments briefly explaining each predicate).
3. Screenshot pictures cut-pasted into the word document showing the execution run of 3 representative test-cases.
4. Answer to each problem must start from a new page.
5. Submit the entire MS Word document containing your solutions on eLearning as a single file.

No collaboration is allowed, all work has to be your own. You can look up your homework solution code, your notes and your textbook. Please do not consult any other source. You may be asked to explain your solution during the grading of the exam.

You are honor bound to not use any unfair means to program the solutions.

Deadline to submit your exam solution: 4pm Thursday, Mar 25th.

Problems

1. **A.** In a farmyard, there are only chickens (that have 2 feet) and rabbits (that have 4 feet). It is known that there are a total of M heads and a total of N feet among all the animals in the farm yard. Your program should calculate and print the number of chickens and number of rabbits in the farmyard as an answer, given M and N. For example:

   ```
   % find_rabbits_and_chickens(M, N, R, C): M is number of heads, N is the number of
   % feet, C is the number of chickens and R is the number of rabbits. Example queries:

   ?- find_rabbits_and_chickens(2, 6, R, C).
      R = 1,  C = 1
   ?- find_rabbits_and_chickens(14, 48, R, C).
       R = 10, C = 4
   ```

   Code this problem in Prolog (please DO NOT use CLP(FD)).

   B. Consider the problem 1 A above. Suppose we only know the total number of feet in the farmyard. Write a program that will compute the total number of rabbits and chickens in the farmyard given just the total number of feet. Assume that the maximum number of animals the farmyard can hold is 40.

```
% find_total_number_of_animals(F, A): F is the total number of feet of these animals in the
% farm; A is the the total number of animals (chickens + rabbits) .
% Example queries:

? find_total_number_of_animals(12, A).

    A = 3;              (0 chickens, 3 rabbits)
    A = 4;               (2 chickens, 2 rabbits)
    A = 5;              (4 chickens, 1 rabbit)
    A = 6;              (6 chickens, 0 rabbit)
    no
```

Code the predicate find_total_number_of_animals/2 in Prolog (please don't use CLP(FD)).

2. Consider the block's world problem in HW#3 (problem is described in Section 14.2 [page 267] of the The Art of Prolog book]). Suppose we are also given the weight of each block as well as its color. This weight information is given as a set of facts, e.g.,

   ```
   weight(a, 2.2, green).          %weight of block a is 2.2 ounces and its green
   weight(b, 6.1, red).            %weight of block b is 6.1 ounces and its red
   etc.
   ```

   Suppose we have the additional constraint that a block X cannot be placed on top of another block Y if the weight of X *exceeds* that of Y by 3 ounces or more. For example, if X weighs 6 ounces and Y weighs 4 ounces, then block X *can be* placed on top of block Y. However, if, say, X weighs 7.5 ounces, and Y weighs 4 ounces, then X *cannot* be placed on top of Y (Y can be placed on top of X though). We have yet another additional constraint that a block cannot be placed on top of another block if the two have the same color.

   Rewrite the block's world program in the textbook incorporating these two additional constraints on weight and color of blocks stated above.

3. Suppose you have 4 bins each containing pennies, nickels, dimes and quarters. Assume that the supply of these coins in each bin is unlimited. Write a predicate (in Prolog) called **calculate/3** that will tell you how many of each type of coins should you pick to make up a given amount, **A**, given a restriction on the *maximum* number, **M,** of coins you can use. Thus, if your program is given the input amount as 75 cents and the limit on the maximum number of coins you can use as 6 (**A** = 75, **M** = 6), then some of the possible solutions are:

   3 quarters

2 quarters, 2 dimes and 1 nickel

2 quarters, 1 dime and 3 nickels

Thus, **calculate/3** takes 2 inputs (**A** and **M**) and produces one output, a four element list that tells us how many each of pennies, nickels, dimes and quarters were used to make up the amount **A** such that the total number of coins used is less than **M**.

4. The mini-Sudoku puzzle is a 2x2 version of the standard 3x3 Sudoku puzzle. Mini-Sudoku only uses digits from 0-3. The idea is to place numbers 0, 1, 2, 3 in a 4x4 table such that each row has unique numbers, each column has unique numbers, and each of the four 2x2 cells (shown with bold lines below) have unique numbers. An example is shown below:

0	1	3	2
2	3	1	0
1	2	0	3
3	0	2	1

Write a Prolog program to generate all possible mini-Sudoku solutions. You should write your program so that it will print a new solution (using the write/1 built-in predicate) each time you hit semi-colon. The solution can be output by simply printing the numbers in each row one line at a time separated by spaces. For example, the above solution will be printed as:

```
?- minisudoku.

    0  1  3  2
    2  3  1  0
    1  2  0  3
    3  0  2  1
    ;                    %hit semi-colon to get a second solution shown below:

    3  0  2  1
    1  2  0  3
    2  3  1  0
    0  1  3  2
    ;

    ....
    ....
```

5. Write a Prolog program called convert/3 that takes a integer decimal number **N**, a base **B** between 1 and 16, and outputs **O,** the number **N** represented in base **B**. You can represent

digits beyond 9, i.e., 10 through 15 (that are needed when you use a base higher than 10) as ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, respectively. You can output **O** as a list of digits. For example:

```
?- convert(5, 1, L).                  %convert decimal number 5 to base 1
   L = [1, 1, 1, 1, 1]
?- convert(5, 2, L).                  %convert decimal number 5 to base 2 (binary)
  L = [1, 0, 1]
?- convert(300, 16, L).               %convert decimal number 50 to base 16 (hexadecimal)
  L = [1, 2, c]
```