ICS 1015 / ICS 5002 Assignment 2016/17

Date due: 20th January 2017

For each question, first write a concise description, in English, of the logic you are going to use to solve the problem. This should then be followed by commented code that matches your description.

/* Use comments to explain what variable names stand for, or what a particular line of code does. */

It is important that your submission includes, for each question, runnable code that can be loaded (or copied and pasted) into the Prolog interpreter and run successfully – so do test it properly before submitting it, and make sure that it is very clear where the code for each problem begins and ends.

Please submit the code as separate sections in your document when uploading it as a single submission to the VLE. If you are unable to upload your submission to the VLE, then please hand in your assignment to the departmental office and ensure that you include the code as one file (with .pl extension) per question on a cd/dvd/usb drive that bears your name.

If you have any questions, please do not hesitate to contact me by email using this address:
peter.xuereb@um.edu.mt

Question 1

A ***deque***, represented as a list, is a double-ended queue data structure that allows the 'pushing' and 'popping' (to push is to add to, to pop is to remove from) of items from either the left or the right side of the list. Write prolog clauses to perform the following deque functions:

a) **leftPush(Item, OldDeque, NewDeque)** that pushes the **Item** on to the left hand side of the **OldDeque** to create the **NewDeque**.

Output should look like this:

```
| ?- leftPush(6,[1,2,3],L).
L = [6,1,2,3]

(ie, L binds to [6,1,2,3]).
```

b) **rightPop (OldDeque, Item, NewDeque)** that pops an **Item** off the right hand side of the **OldDeque** and returns the **NewDeque** now missing that item.

```
e.g., rightPop([6,1,2,3],I,L). binds I to 3 and L to [6,1,2].
```

c) similar to the functions above, **leftPop** and **rightPush**, which pop an item off the left hand side of the deque, and push an item onto the right hand side of the deque, respectively.

d) **checkEmpty(Deque)** returns True if **Deque** empty and False if not.

```
e.g., checkEmpty([6,1,2,3]). returns no.
```

e) **dequeSize(Deque,Size)** that returns, as **Size**, the number of items in the **Deque**.

```
e.g., dequeSize([6,1,2,3],S). binds S to 4.
```

Question 2

One way to solve two simultaneous equations is to eliminate one of the unknown coefficients. After solving the value of one coefficient, the known value can be used in any of the two equations to solve the second unknown coefficient. The equations can be represented as Prolog facts -- for example, *eq(2,1,4)* represents the equation *2x + y = 4* and *eq (1,-1,-1)* represents *x – y = -1*.

Write Prolog clauses to perform the following functions to solve simultaneous equations:

a) **extractCoef(Coef, Equation, Value)** that returns the value of a particular coefficient from an equation.
e.g. `extractCoef(x,eq(2,1,4),V). binds V to 2.`

b) **sameCoefSign(Coef1, Coef2)** returns True if the coefficients have the same sign, otherwise returns False if the coefficients have different signs.
e.g. `sameCoefSign(-2,-1). returns True`

`sameCoefSign(-2,1). returns False.`

c) **raiseEq(Multiple, OrigEq, FinalEq)** that raises the Original Equation with the Multiple value to return the Final Equation.
e.g. `raiseEq(2,eq(1,-1,-1),F). binds F to eq(2,-2,-2).`

d) **solveSim(Eq1, Eq2, Xvalue, Yvalue)** that returns the value of coefficients X and Y after simultaneously solving Equation1 and Equation 2.
e.g. `solveSim(eq(2,1,4),eq(1,-1,-1),X,Y). binds X to 1 and Y to 2.`

e) **checkSol(Eq1,Eq2,Xvalue,Yvalue)** that returns True if the values of X and Y supplied satisfy both equations Eq1 and Eq2.
e.g. `checkSol(eq(2,1,4),eq(1,-1,-1),1,2). returns yes.`

Question 3

A ratio is a comparison between different portions, and is usually used to distribute an amount according to this comparison. For example, the ratio 3:1:2 can be represented as a Prolog fact as ratio([3,1,2]).

Write Prolog predicates to work out the following ratio functions:

a) **sumOfRatios(Ratio, Sum)** that binds **Sum** to the total of the different ratios represented in the fact **Ratio**.
e.g. `sumOfRatios(ratio([3,1,2]),S). binds S to 6.`

b) **reduceRatio(OriginalRatio, FinalRatio)** that reduces the **OriginalRatio** to its lowest term and binds it to the **FinalRatio** (ie, 30:10:20 is the same as 3:1:2).
e.g. `reduceRatio(ratio([30,10,20]),R). binds R to ratio([3,1,2]).`

c) **divideRatio(Amount, Ratio, Parts)** binds **Parts** with a list of values that originated from the **Amount** after that it was divided according to the specified **Ratio**.
e.g. `divideRatio(54,ratio([30,10,20]),P). binds P to [27,9,18].`

Hints: You will need to compute the GCD of several numbers for part (b).

Question 4.

A set is a collection of elements, rather like a list, with the difference that it does not make sense to ask where or how many times something is a member of a set. The set {4,6,2} is thus the same as the set {6,2,4}.

Write Prolog predicates to perform the following set functions:

a) **mem_of(M, S)** that returns True if the element M is a member of the set S.
e.g. `mem_of(6,[2,6,3,4]). returns yes`

`mem_of(6,[2,5,3,4]). returns no.`

b) **subset_of(S1, S2)** that returns True if the set S1 is a subset of the set S2 (ie, all elements in S1 can be found in S2).
e.g. `subset_of([2,4],[2,6,3,4]). returns yes.`

c) **intersect(S1, S2, S3)** that returns the set S3, where all elements of S3 are present in both S1 and S2.
e.g. `intersect([20,4,13,11,24],[33,2,4,11,20,68],S). binds S to [20,4,11].`

d) **unite(S1, S2, S3)** that returns the set S3, where all elements in S1 and S2 are present in S3.
e.g. `unite([5,2,6,3,4],[1,5,7,2,4],S). binds S to [6,3,1,5,7,2,4].`