ICS 1015 Assignment 2017/18

Date due: 16th January 2018

Your submission to the VLE should consist of several parts, namely:

a) a single document which contains an explanation and the relevant code for all questions, each question in a separate section of the document;
b) a separate file (with .pl extension) with commented code for each question answered.

In your document (see (a) above), for each question, first write a concise description, in English, of your understanding of the problem and the logic you are going to use to solve it, and follow this by commented code that matches your description. Very important to note is that your ***explanations*** of the code you have written ***are as important as the code itself*** – so ensure that these are well written and clear!

Your submission must include, for each question, runnable code (see (b) above) that can be loaded (or copied and pasted) into the Prolog interpreter/compiler 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 put your name in comments at the top of each file.

/* … and use comments to explain what variable names stand for or what particular lines of code do! */

If you are unable to upload your submission to the VLE, then please contact the departmental office to see how best to submit your assignment.

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

Question 1

Write a program that has 'in mind' a secret word (hello), where the user has to guess this word, letter by letter.

The program asks for the first letter with the prompt "Guess the first letter". If you guess correctly, it will prompt you for the next letter with "OK! Letter #<n>", where n is the position of the next letter to be guessed. If not, it will prompt you to guess again with "Sorry -- try again! Letter #<n>".

When you have come to the end of the word, it will print "Congratulations – the word is hello".

Make use of the prolog predicates **repeat/0** and **->/2** in your answer.

Your output should look like this:

```
| ?- start.
Guess first letter|: h.
OK. Letter #2|: e.
OK. Letter #3|: l.
OK. Letter #4|: l.
OK. Letter #5|: o.
OK. Congratulations! The word is [h,e,l,l,o]
yes

| ?- start.
Guess first letter|: p.
Guess first letter|: l.
Guess first letter|: h.
OK. Letter #2|: l.
Fail. Try again! Letter #2|: e.
OK. Letter #3|: l.
OK. Letter #4|: p.
Fail. Try again! Letter #4|: l.
OK. Letter #5|: a.
Fail. Try again! Letter #5|: o.
OK. Congratulations! The word is [h,e,l,l,o]
yes

| ?-
```

[15 marks]

Question 2

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).

[20 marks]

Question 3

a) Write a prolog predicate **permute/2** that takes a list of items and returns its permutations.

Your output should look something like this:

```
| ?- permute([1,2,3],R).
R = [1,2,3] ;

R = [1,3,2] ;

R = [2,1,3] ;

R = [2,3,1] ;

R = [3,1,2] ;

R = [3,2,1] ;

no
```

b) Explain the way your algorithm works in detail.

c) An anagram is word or phrase formed by rearranging the letters of a different word or phrase using all the original letters exactly once. Use **permute/2** and the prolog predicate **findall/3** to write a new predicate **anagram/2**, which computes all anagrams of a given list and returns the results in a single list of lists.

Your output should look something like this:

```
| ?- anagram([a,b,c],A).
A = [[a,b,c],[b,a,c],[b,c,a],[a,c,b],[c,a,b],[c,b,a]]
```

d) Write a prolog predicate **sorted/1** that takes a list of numbers and succeeds if they are in order.

Your output should look like this:

```
| ?- sorted([1,2,3,4,5]).
yes
```

e) What is a Naïve Sort? Use your answers in (a) and (d) above to implement a Naïve Sort in prolog.

Your output should look like this:

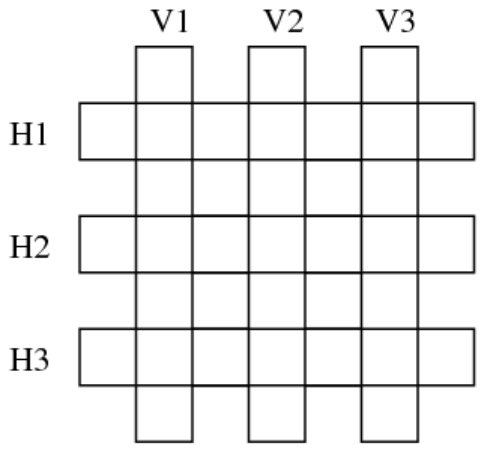
```
| ?- nSort([2,5,3,1,5,7,8,7,9],L).
L = [1,2,3,5,5,7,7,8,9]
```

[30 marks]

Question 4a

Here are six Italian words: astante , astoria , baratto , cobalto , pistola , statale.

They are to be arranged, crossword puzzle fashion, in the following grid:



The following knowledge base represents a lexicon containing these words:

```
word(astante, a,s,t,a,n,t,e).
word(astoria,  a,s,t,o,r,i,a).
word(baratto, b,a,r,a,t,t,o).
word(cobalto, c,o,b,a,l,t,o).
word(pistola,  p,i,s,t,o,l,a).
word(statale,  s,t,a,t,a,l,e).
```

Write a predicate **crossword(V1,V2,V3,H1,H2,H3)** that gives solutions as to how to fill in the grid. The first three arguments, **V1**, **V2**, **V3**, represent the vertical words from left to right, and the last three arguments, **H1**, **H2**, **H3**, the horizontal words from top to bottom.

(Hint: pay special attention to the intersections of the verticals and horizontals).

[15 marks]

Question 4b

The Towers of Hanoi (https://en.wikipedia.org/wiki/Tower_of_Hanoi) is a classic mathematical puzzle that nicely illustrates the use of recursion to solve problems. The puzzle consists of three poles, left, centre and right, and a stack of discs placed one on top of the other in descending order of size, from largest on the base, to smallest, at the top of the stack. The objective of the puzzle is to move the stack of discs from the left pole over to the right pole, one at a time, so that they end up in the original order on that pole. The pole in the centre is used as a temporary resting place for the discs, but at no time is a larger disc allowed to be on top of a smaller one.

Outline an algorithm to solve the problem.

Now write a prolog program – or find one on the internet if you wish – which, when executed, will solve the problem for any size of stack.

If you decide to find the program on the internet, then select the best example which, in your view, most elegantly illustrates the solution to this problem, and explain why.

If you decide to write the program yourself, explain how the algorithm that you just outlined maps to the program you have written. You may choose any way you wish to represent the towers, but a nice approach could be to represent these as three lists (or a list of 3 lists), and print the intermediate results in a way that is similar to the following (in this case 1 represents the smallest disc, 4 the largest disc):

Size=4

L = [1,2,3,4] C = [] R = []
L = [2,3,4] C = [1] R = []
L = [3,4] C = [1] R = [2]
L = [3,4] C = [] R = [1,2]
L = [4] C = [3] R = [1,2]
L = [1,4] C = [3] R = [2]
L = [1,4] C = [2,3] R = []
L = [4] C = [1,2,3] R = []
L = [] C = [1,2,3] R = [4]
L = [1] C = [2,3] R = [4]
L = [1] C = [3] R = [2,4]
L = [] C = [1,3] R = [2,4]
L = [2] C = [1,3] R = [4]
L = [1,2] C = [3] R = [4]
L = [1,2] C = [] R = [3,4]
L = [2] C = [1] R = [3,4]
L = [] C = [1] R = [2,3,4]
L = [] C = [] R = [1,2,3,4]

[20 marks]