COMP 329 KNOWLEDGE SYSTEMS
ASSIGNMENT 2 – Understanding Prolog Programming

Due by 12midnight Friday October 31 (Week 11)

Submission Guidelines

This assignment will contribute 15% towards your final grade. It consists of 3 questions (totalling 15 marks) with up to 5 bonus marks, depending on your choice for the last (puzzle) question.

You must submit all of your Prolog code in a single file called `a2.pl` using **Blackboard/WebCT** by the due date.

Your must also submit a hardcopy of your program together with a sample output showing that your code works; this means that you should test your program with a number of test cases including the boundary conditions. You should place the hardcopy submissions in the the COMP329 assignment boxes in the first floor of E6A by the due date.

Late submissions will NOT be accepted. No extensions will be given for any of the assignments (except in case of illness or misadventure).

Note: To perform satisfactorily in the assignment component of the course, at least one of the two assignments must be attempted AND you must get at least 15% jointly from all the assignments that you submit.

Question 1 (4 marks)

You are given the following Prolog program:

```
appears_before(X, Y, [X|T]) :- member(Y, T).           % ab.1
appears_before(X, Y, [_|T]) :- appears_before(X, Y, T). % ab.2

member(X, [X|_]).                                       % m.1
member(X, [_|T]) :- member(X, T).                       % m.2
```

And the goal `?- appears_before(X, 3, [1,2,3,3]).` Draw the corresponding whole SLD-tree (or proof tree) for the given program and the goal. Annotate the tree with the substitutions and the corresponding clause numbers at each resolution step.

Attach the proof tree to your hardcopy submission. Hand-drawn trees are acceptable, but they must be neat.

Question 2 (6 marks)

A bag (or a multiset) is a collection of items which may contain more than one occurrence of each item, eg., {a, b, b, c}. The order of elements in a bag does not matter, for example, the bag {a, b, b, c} is the same as the bag {b, a, c, b}. In Prolog bags can be represented using lists, eg., here is a bag of constants: `[a, b, b, c]`.

A difficulty arises with the list representation of bags in the sense that we would like the two bags to be **equal** if they have the same elements the same number of times. For example, it should be the case that `[a, b, b, c] = [b, a, c, b]`. However, these two lists are not considered to be equal in Prolog (as they cannot be unified). So, a special equality predicate for bags would be required.

To overcome this difficulty, we can represent bags as a list of pairs where each pair consists of a bag element together with the number of its occurrences in the bag. For the above example, we would have the following representation: `[[a,1],[b,2],[c,1]]`. Each bag element would occur in only one pair in a given bag; elements with zero occurrence are not listed in the bag.

Note that in this representation the bag elements are sorted in increasing order (sorted to the standard order of terms in Prolog) so that there is a **unique** (or canonical) list of pairs for each given bag. Therefore we do not have multiple representations for the same bag, and two bags are considered to be equal if they can simply be unified.

Write and test Prolog procedures for the following predicates (we assume the canonical list representation of bags unless otherwise stated):

(a) (3 marks) `bagify(L, B)` is true if `L` is a list of items (a bag) and `B` is the canonical list of pairs representing `L`.

Example:

```
?- bagify([b,a,c,b,b,b], B).
B = [[a,1], [b,4], [c,1]]
Yes
```

(b) (1 mark) `bag_intersection(B1, B2, B)` is true if bag `B` consists of the elements thar are in both bags `B1` and `B2` and the number of occurrences of each element in `B` is equal to the minimum of the numbers of its occurrences in `B1` and `B2`.

Example:

```
?- bag_intersection([[a,3], [b,1], [c,1]], [[a,2], [c,2]], B).
B = [[a,2], [c,1]]
Yes
```

(c) (1 mark) `bag_union(B1, B2, B)` is true if bag `B` consists of the elements which are in bag `B1` or `B2` (or both) and the number of occurrences of each element in `B` is the sum of the numbers of its occurrences in bags `B1` and `B2`.

Example:

```
?- bag_union([[a,3], [b,1], [c,1]], [[a,2], [c,2]], B).
B = [[a,5], [b,1], [c,3]]
Yes
```

(d) (1 mark) `bag_difference(B1, B2, B)` is true if bag `B` consists of the elements which are in bag `B1` but not in `B2` and the number of occurrences of each element in `B` is the number of its occurrences in bag `B1` minus that in bag `B2`.

Example:

```
?- bag_difference([[a,3], [b,1], [c,1]], [[a,2], [c,2]], B).
B = [[a,1], [b,1]]
Yes
```

Question 3 (5-10 marks)

This question involves solving a particular logic puzzle. There are three puzzles to choose from, and each puzzle has a different weight depending on its difficulty level as shown below. **Attempt only one puzzle**; you will not gain any extra marks if you attempt more than one puzzle.

Puzzle	Worth
open_the_box.puz	5 marks
prefect_duty.puz	7 marks
star_chart.puz	10 marks

Your program code should include, as comments, a succinct explanation of the strategy you adopt for the solution of the puzzle. Also make sure to annotate parts of your code that correspond to each clue; the best way to do this is to embed your code into the puzzle description. Do not reorder the clues (unless it is absolutely necessary).

Demonstrate that your program works and that it produces **a unique ground solution** for the puzzle (there should be no variables in it). The solution should be produced **only once** (test this by typing a semi-colon after Prolog displays the first solution).

Full marks will not be gained by just having the correct solution. Efficiency, conciseness and readability will also be taken into account.

Note: The descriptions of the above puzzles and the guidelines for solving puzzles are available on the COMP329 unit website http://www.comp.mq.edu.au/units/comp329/Puzzles/puzindex.html. There are also sample solutions to a selected set of puzzles.