CT331 Assignment 3

Declarative Programming with Prolog

Due Date: TBC

Introduction

Please submit a PDF file clearly showing the following:

1. The assignment name ("CT331 Assignment 3")
2. Your name and student ID
3. Each question title ("Question 1"), in order, displayed clearly above your answer.
4. For each question:
    a. A copy of your code (not a screenshot) and a screenshot of your output, including adequate tests to demonstrate that your code works.
    b. Any comment you may have, or whatever textual answer the question requires.

Using the lab computers:

SWI Prolog should be installed on the lab computers. SWI Prolog is also available to download from http://www.swi-prolog.org/

Question 1

Consider the following prolog facts:

```
takes(tom, ct331).
takes(mary, ct331).
takes(joe, ct331).
takes(tom, ct345).
takes(mary, ct345).
instructs(bob, ct331).
instructs(ann, ct345).
```

1. Write a prolog rule called ‘teaches’ that returns true if a given instructor teaches a given student.
2. Write a prolog query that uses the ‘teaches’ rule to show all students instructed by bob.
3. Write a prolog query that uses the ‘teaches’ rule to show all instructors that instruct mary.
4. What is the result of the query:
   teaches(ann, joe).
   Why is this the case?
5. Write a prolog rule called ‘classmates’ that returns true if two students take the same course. Demonstrate with suitable queries that this rule works as described. [1 mark each]

Question 2

1. Using the “=” sign and the prolog list syntax to explicitly unify variables with parts of a list, write a prolog query that displays the head and tail of the list [1 ,2,3].
2. Similarly, use a nested list to display the head of the list, the head of the tail of the list and the tail of the tail of the list [1 ,2,3,4,5].
3. Write a prolog rule ‘contains1’ that returns true if a given element is the first element of a given list.
4. Write a prolog rule ‘contains2’ that returns true if a given list is the same as the tail of another given list.
5. Write a prolog query using ‘contains1’ to display the first element of a given list.

[1 mark each]

Question 3

Write a set of prolog facts and rules called `isNotElementInList` that determine if a given element is **not** in a given list. The element must be the first argument and the list must be the second.

Eg: `isNotElementInList(El, List) …`

Your code should produce the following results:

```
?- isNotElementInList(1, []).
True.
?- isNotElementInList(1, [1]).

False.

?- isNotElementInList(1, [2]).
True.
?- isNotElementInList(2, [1, 2, 3]).
False.
?- isNotElementInList(7, [1, 2, 9, 4, 5]).

True. (5 marks)
```

Question 4

Write a set of prolog facts and rules called `mergeLists` that merges (concatenates / appends) three lists. The given lists must be the first three arguments, and the merged list must be the fourth argument. **Do not use the built in append rule.**

Eg: `mergeLists(List1, List2, List3, Merged) …`

Your code should produce the following results:

```
?- mergeLists([7],[1,2,3],[6,7,8], X).
X = [7,1,2,3,6,7,8].
?- mergeLists([2], [1], [0], X).
X = [2, 1, 0].
?- mergeLists([1], [], [], X).
X = [1]. (5 marks)
```

Question 5

Write a set of prolog facts and rules called `reverseList` that reverses a given list. The given list must be the first argument and the reversed list must be the thrird

Eg: `reverseList(List, ReversedList) …`

Your code should produce the following results:

```
?- reverseList([1,2,3], X).
X = [3,2,1].
?- reverseList([1], X).
X = [1].
?- reverseList([], X).
X = []. (5 marks)
```

Question 6

Write a set of prolog facts and rules called `insertInOrder` that inserts an element into its correct position in a given list. The given element must be the first argument, the given list must be the second and the final list with the inserted element must be the third argument. You can assume that the 2nd argument is sorted in ascending order,

Eg: `insertInOrder(El, List, NewList) …`

Your code should produce the following results:

```
?- insertInOrder(7,[1,2,3], X).
X = [1,2,3,7].
?- insertInOrder(2, [3], X).
X = [2, 3].
?- insertInOrder(1, [], X).
X = [1]. (5 marks)
```