Large Assignment #2: Prolog
Due: Friday, 15 November 2024 by 11:59 PM

Instructions.
Implement the predicates below, following all the guidelines, in a file called `la2.pl` and submit it on lectura using the following command:

```
turnin csc372la2 la2.pl
```

General Guidelines.

- You are only allowed to use the Prolog features discussed in class or provided in the notes on D2L.
- You should use pattern matching and minimize the use of unnecessary helper predicates and unnecessary rules and goals.
- If any predicates are incomplete and/or do not compile, comment them out.
- Double-check the spelling of the predicate names and the types.
- Put the number of the question in a comment above the predicate implementation.
- Get rid of all Singleton warnings.
- When testing, use the sample queries. As long as you get the correct results, the order does not matter. It also does not matter if you get duplicates or not. What DOES matter is getting the correct results, not getting incorrect results, and not getting stuck in any infinite loops or crashing the program.
- The *company.txt* file is provided so you can more easily test your predicates having to do with management (questions 1-5).
- In some cases, the queries have so many possible answers that I do not print them all.

Academic Integrity.
If you are found to have violated the academic integrity policy, you will be reported to the Dean of Students office and the recommended sanction will be a failing grade in the course.

Grading.

These may be graded for accuracy and style, and will be tested on lectura, so make sure they work on lectura before you submit. Questions 1-10 are worth 4 points each. Questions 11-14 are worth 10 points each.

Questions 1 - 10

Number	Predicate	Meaning	Sample Queries

1	`teammate(X,Y)`	`X and Y are on the same team, managed by the same person`	`?- teammate(juan,X).`
			`X = george ;`
			`X = harry ;`
			`X = inez ;`
			`X = kevin.`
			`?- teammate(X,leon).`
			`false.`
			`?- teammate(X,Y).`
			`X = bob,`
			`Y = carla ;`
			`X = bob,`
			`Y = dave ;`
			`X = bob,`
			`Y = elsa .`
			…

2	`bossOfBoss(X,Y)`	`X is the boss of Y's boss`	`?- bossOfBoss(X,bob).`
			`false.`
			`?- bossOfBoss(X,inez).`
			`X = alice.`
			`?- bossOfBoss(alice,X).`
			`X = george ;`
			`X = harry ;`

			X = inez ; X = juan ; X = kevin ; X = stephen ; X = tracy ; X = eddie ; false. ?- bossOfBoss(X,Y). X = alice, Y = george ; X = alice, Y = harry ; X = alice, Y = inez ; X = alice, Y = juan . …
3	superior(X,Y)	X is a superior to Y (i.e. higher up in the chain of command)	?- superior(X,alice). false. ?- superior(X,bob). X = alice . ?- superior(X,eddie). X = elsa ; X = alice ; false. ?- superior(eddie,X). X = francine ;

			X = ginny ; X = hans ; X = iris ; X = jack ; false.
4	sameRank(X,Y)	X and Y are at the same rank in the management chain	?- sameRank(eddie,X). X = eddie ; X = george ; X = george ; X = harry ; X = harry . … ?- sameRank(X,bob). X = bob ; X = carla ; X = dave ; X = elsa ; X = frank . … ?- sameRank(X,Y). X = Y ; X = bob, Y = carla ; X = bob, Y = dave ; X = bob, Y = elsa ;

			X = bob, Y = frank . …
5	chainOfCommand(X) 	X is a list indicating a chain of command from highest to lowest rank	?- chainOfCommand(X). X = [_] ; X = [alice, bob] ; X = [alice, bob, george] ; X = [alice, bob, george, keeley] ; X = [alice, bob, harry] ; X = [alice, bob, inez] . …
6	last(X,Y)	X is the last element in list Y.	?- last(X,[]). false. ?- last(X,[4,1,5,6]). X = 6 ; false. ?- last(4,X). X = [4] ; X = [_, 4] ; X = [_, _, 4] . …
7	merge(X,Y,Z)	Z is sorted list X merged with sorted list Y; do not use append or sort	?- merge([1,3,5],[2,4,6,8],Z). Z = [1, 2, 3, 4, 5, 6, 8] ; false.
8	repeat(X,N,Z)	Z is list X repeated N times.	?- repeat([1,2,3],1,Z).

			Z = [1, 2, 3] . ?- repeat([1,2,3],2,Z). Z = [1, 2, 3, 1, 2, 3] . ?- repeat([1,2,3],3,Z). Z = [1, 2, 3, 1, 2, 3, 1, 2, 3] . ?- repeat(X,2,[1,2,3,1,2,3]). X = [1, 2, 3] .
9	sublist(X, I, J, Y)	Y is a sublist of X starting at index I (inclusive) and ending at index J (exclusive); indexing starts at 0; you can assume that the indexes given will be appropriate for the list	?- sublist([1, 2, 3, 4, 5], 2, 4, X). X = [3, 4] . ?- sublist(X, 0, 3, [1, 2, 3]). X = [1, 2, 3|_] . ?- sublist([8, 9, 10], 0, 1, X). X = [8] .
10	insert(X, Y, Z)	Assume Y is a sorted list; Z is Y with X inserted into it so that it is still sorted.	?- insert(3, [], X). X = [3] . ?- insert(3, [0, 2, 4, 6], X). X = [0, 2, 3, 4, 6] . ?- insert(8, [1, 2, 3, 8, 8], X). X = [1, 2, 3, 8, 8, 8] .

Question 11. The Sudoku Problem

This problem is similar to the 8-Queens problem, but instead we are dealing with a mini version of a Sudoku puzzle. The board for this problem is a 4×4 square. A solution to the puzzle is a placement of 1's, 2's, 3's, and 4's so that

- every row has a 1, a 2, a 3, and a 4
- every column has a 1, a 2, a 3, and a 4
- each of the four quarters of the grid $(2 \times 2\ squares)$ has a 1, a 2, a 3, and a 4.
- no row has two of the same number
- no column has two of the same number
- none of the four quarters of the grid has two of the same number

Example (the four quarters are shown by shading):

1	2	3	4
3	4	1	2
2	3	4	1
4	1	2	3

Start with the following fact that represents the locations in the board:
`board([1/1, 1/2, 1/3, 1/4, 2/1, 2/2, 2/3, 2/4, 3/1, 3/2, 3/3, 3/4, 4/1, 4/2, 4/3, 4/4]).`

You will probably need several predicates to solve this problem, but the one that will be used in the queries is:

`fullPlacement(X): X is a possible placement for all the numbers in the grid (X is a list of lists).`

Recommendations:

- Define `sameRow`, `sameCol`, and `sameSection` predicates. For the `sameSection` predicate, you can use a built in predicate called `ceiling`, which can be evaluated using `=:=` or `is` operator.
- Define a `legalloc(X, Y)` predicate, which means that `X` is a legal location with respect to the list of other locations `Y`.
- Define a `legal(X)` predicate, which means that `X` is a legal placement for a specific value (a list of locations that will work for that value).
- I also used a `removeAll(X, Y, Z)` predicate where `X`, `Y`, and `Z` are lists and `Z` is `Y` with `X` removed.
- I also used a `subset(X, Y)` predicate where `X` is a subset of `Y`.

Example Queries.

```
?- fullPlacement([Ones, Twos, Threes, Fours]).
Ones = [1/1, 2/3, 3/2, 4/4],
Twos = [1/2, 2/4, 3/1, 4/3],
Threes = [1/3, 2/1, 3/4, 4/2],
Fours = [1/4, 2/2, 3/3, 4/1] .

?- Ones = [1/1, 2/3, 3/4, 4/2], fullPlacement([Ones, Twos, Threes, Fours]).
Ones = [1/1, 2/3, 3/4, 4/2],
Twos = [1/2, 2/4, 3/1, 4/3],
Threes = [1/3, 2/1, 3/2, 4/4],
Fours = [1/4, 2/2, 3/3, 4/1] .
```

Note that these queries may have multiple answers. I am only printing out one of the possible answers.

Question 12.
In the traveling salesperson problem, we start with a graph where vertices are cities and the edges are the distances between those cities. The goal is to find a route that

- starts and ends at the same city
- visits every city exactly once
- minimizes the total distance traveled

Since finding a route that is actually optimal is difficult, this version of the problem specifically looks for a route that

- starts and ends at the same city
- visits every city exactly once
- has a total distance that is less than or equal to a maximum distance parameter

Note that this variation is an NP-Complete problem.

Start with the following database information to represent the graph.

```
fullGraph([albany, annapolis, atlanta, austin, baton_rouge, boston]).

dist(albany, annapolis, 469).
dist(albany, atlanta, 1356).
dist(albany, austin, 2538).
dist(albany, baton_rouge, 2056).
dist(albany, boston, 224).
dist(atlanta, annapolis, 915).
dist(atlanta, austin, 1318).
dist(atlanta, baton_rouge, 736).
dist(atlanta, boston, 1507).
dist(annapolis, austin, 2168).
dist(annapolis, baton_rouge, 1639).
```

```
dist(annapolis, boston, 593).
dist(austin, baton_rouge, 634).
dist(austin, boston, 2729).
dist(baton_rouge, boston, 2225).
dist(tucson, atlanta, 2478).
dist(tucson, albany, 2526).
dist(tucson, annapolis, 3186).
dist(tucson, baton_rouge, 1887).
dist(tucson, boston, 3669).
dist(tucson, austin, 1275).
```

Like previous problems, you will probably need to write several predicates, but the required one that will be used in queries is:

`tsp(G, M, C, D)`: `C` is a cycle in `G` that visits every city exactly once and whose total distance `D` is less than or equal to `M`.

Example Query:

```
?- tsp(G, 5700, C, D).
G = [albany, annapolis, atlanta, austin, baton_rouge, boston],
C = [boston, albany, austin, baton_rouge, atlanta, annapolis, boston],
D = 5640 .
```

Note that there are multiple possibilities for this query. I just printed one to show the format.

Recommendations:

- Write a `distance(X, Y, D)` predicate to handle the fact that distances between cities go both ways.
- Write (or use from a previous assignment) a `last(X, Y)` predicate where `Y` is the last element of list `X`.

- Write a `cycle(X, Y)` predicate where `X` is a spanning cycle of `Y` (i.e. it starts and ends at the same place and hits every element).
- Write a `totalDistance(X, Y)` predicate where `Y` is the total distance covered by path/cycle `X`.
- I also used an `eqLists(X, Y)` predicate that is true if `X` and `Y` have the same elements (but not necessarily in the same order).

Question 13.
The K-Clique Problem.
A clique in a graph is a subgraph in which all the vertices are connected to each other. For example, if you draw Graph 1, you will see several cliques, including: {a,b,e} and {b,c,d,e}.
In the k-clique problem, you need to write a predicate that finds a clique of size K in a given graph (if one exists). The predicate below is required for the queries, but you should break down the problem into other predicates as necessary.

```
kclique(G,C,K): C is a clique of size K in graph G
```

Example Queries:

```
?- G=[a,b,c,d,e,f,g,h], kclique(G,C,4).
G = [a, b, c, d, e, f, g, h],
C = [b, c, d, e] .
?- G=[a,b,c,d,e,f,g,h], kclique(G,C,3).
G = [a, b, c, d, e, f, g, h],
C = [a, b, e] .
?- G=[a,b,c,d,e,f,g,h], kclique(G,C,5).
false.
```

Question 14.

The Dominating Set Problem.

Let G be a graph. Let D be a subset of the vertices in G. D is a dominating set of G if every vertex in G is either in D or has a neighbor in D. In this problem, you need to write a predicate (with helpers) that determines a dominating set of a graph G of a given size. The required predicate is described below.

```
dominatingSet(G,D,S): D is a dominating set of graph G of size S
```

Example Queries:

```
?- G=[a,b,c,d,e,f,g,h], dominatingSet(G,D,2).
G = [a, b, c, d, e, f, g, h],
D = [b, f] .
?- G=[a,b,c,d,e,f,g,h], dominatingSet(G,D,1).
false.
?- G=[a,b,c,d,e,f,g,h], dominatingSet(G,D,4).
G = [a, b, c, d, e, f, g, h],
D = [a, b, c, f] .
```