CS-4337.0u1
Prolog Programming Assignment
Due: August 3, 2015 11:59PM

Description

Define and test the Prolog predicates described below. Each of your predicates ***must*** have the same name and signature as the examples below. In Prolog, predicate profiles are indicated with the number of parameters that they take, e.g. `sort/2` is a predicate named "`sort`" that takes 2 parameters.

Your predicates must behave properly on all instances of valid input types.

Submission

Your submission should consist of a single source code text file that includes all facts, predicate definitions, propositions, and rules. Your file must be named ***`your_net_id`*`.pro`**, then subsequently archived using **`zip`**, **`gzip`**, or **`tar`** and named ***`your_net_id`*`.archive_extension`**. For example, **`cid021000.zip`** or **`cid021000.tar`**.

You may find additional Prolog language help at the following links:

- SWI-Prolog manual
- SWI-Prolog documentation
- Learn Prolog Now!
- http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/contents.html

1) Multiple of 3 [10 points]

Define a predicate **`multof3/1`** that determines whether an integer is a multiple of 3. A user should be able to enter the predicate with an integer, e.g. **`multof3(42)`**) and evaluate to either **`true`** or **`false`**. If the given parameter is not an integer, your predicate should display the message "**`ERROR: The given parameter is not an integer`**".

```
?- multof3(171).
true.

?- multof3(100).
false.

?- multof3(0).
true.

?- multof3(4.2).
ERROR: The given parameter is not an integer

?- multof3(-9).
true.
```

2) Minimum [10 points]

Define a predicate **`minim/2`** with the signature **`minim(List, Min)`** where **`Min`** is the minimum valued element in some list of numbers, **`List`**. If one more elements of **`List`** is not a number, then the predicate should display the following for the first encounter of a non-number element, "**`ERROR: "`*`element`*`" is not a number.`**".

```
?- minim([7,5,3,6,-3,9], Min).
Min = -6

?- minim([512], Min).
Min = 512

?- minim([12,2,b,7], Min).
ERROR: "b" is not a number.
```

3) Prime [10 points]

Write a set of rules to determine whether a number is prime. A user should be able to enter the predicate: **`is_prime(10)`** (for example) and get a **`true`** or **`false`**. Your predicate should have the signature **`is_prime(SomeInteger)`**. You may assume that the parameter is an integer.

Comments/Hints:

- The base cases are that 1 and 2 are prime (Technically, 1 isn't prime, but you may assume that it is.).
- Depending on your prime test strategy, you may want to have a helper rule that tells you whether a number evenly divides another and handles the recursion. You'll want to create a helper predicate, **`divisible/2`**, that has the signature **`divisible(X,Y)`** and determines whether X is evenly divisible by Y, i.e. X mod Y = 0.

```
?- is_prime(17).
true.

?- is_prime(21).
false.
```

4) Segregate [10 points]

Define a predicate **`segregate/3`** whose first parameters is a list of integers and the next two parameters are a list of the even member of the first list and odd members of the first list, respectively. The odd and even lists should have the numbers *in the same order* that they appear in the first parameter list. Your predicate should have the signature **`segregate(List, Even, Odd)`**.

```
?- segregate([8, 7, 6, 5, 4, 3], Even, Odd).
Even = [8, 6, 4]
Odd = [7, 5, 3]

?- segregate([7, 2, 3, 5, 8], Even, Odd).
Even = [2, 8]
Odd = [7, 3, 5]
```

5) Sum List [10 points]

Define a predicate **`sum_list/2`** that takes a list of numbers as a first parameter and determines the sum of all of the list elements. Your predicate should have the signature **`sum_list(List, Sum)`**.

```
?- sum_list([4, 3], Sum).
Sum = 7.

?- sum_list([6, 2, 5, 10], Sum).
Sum = 23.

?- sum_list([], Sum).
Sum = 0.
```

6) Product List [10 points]

Define a predicate **`prod_list/2`** that takes a list of numbers as a first parameter and determines the product of all of the list elements. Your predicate should have the signature **`prod_list(List, Product)`**.

```
?- prod_list([4, 3], Product).
Product = 12.

?- prod_list([6, 2, 5, 10], Product).
Product = 600.

?- prod_list([], Product).
Product = 0.
```

7) Bookends [10 points]

Design a predicate **`bookends/3`** that tests if the first list parameter is a prefix of the third and the second is a suffix of the third. Note that the lists in the first and second parameters may overlap.

```
?- bookends([1],[3,4,5],[1,2,3,4,5]).
true.

?- bookends([],[4],[1,2,3,4]).
true.

?- bookends([1,2,3],[3,4],[1,2,3,4]).
true.

?- bookends([1],[2,3],[1,2,3,4]).
false.
```

8) Subslice [10 points]

Design a predicate **`subslice/2`** that tests if the first list parameter is a contiguous series of elements anywhere within in the second list parameter. Your predicate definition must be recursive.

```
?- subslice([2,3,4],[1,2,3,4]).
true.

?- subslice([3],[1,2,4]).
false.

?- subslice([],[1,2,4]).
true.

?- subslice([1,2,4],[]).
false.
```

9) Graph [10 points]

Design two predicates **`path/2`** and **`cycle/1`** that determine structures within a graph whose *directed* edges are encoded with instances of **`edge/2`**. For example, **`path(x,y)`** should evaluate to **`true`** if a path exists from vertex **`x`** to vertex **`y`**, and **`false`** otherwise. And **`cycle(x)`** should evaluate to **`true`** if a cycle exists which includes vertex **`x`**.

Note: All edges are directional

```
Knowledge Base
edge(a,b).
edge(b,c).
edge(c,d).
edge(d,a).
edge(d,e).
edge(b,a).

?- path(b,d)
true.

?- path(e,b).
false.

?- path(c,a).
true.

?- cycle(b).
true.

?- cycle(e).
false.
```

10) Clue [10 points]

Four guests (Colonel Mustard, Professor Plum, Miss Scarlett, Ms. Green) attend a dinner party at the home of Mr. Boddy. Suddenly, the lights go out! When they come back, Mr Boddy lies dead in the middle of the table. Everyone is a suspect.

Upon further examination, the following facts come to light:

- Mr Boddy was having an affair with Ms. Green.
- Professor Plum is married to Ms. Green.
- Mr. Boddy was very rich.
- Colonel Mustard is very greedy.
- Miss Scarlett was also having an affair with Mr. Boddy.

There are two possible motives for the murder:

- Hatred: Someone hates someone else if that other person is having an affair with his/her spouse.
- Greed: Someone is willing to commit murder if they are greedy and not rich, *and* the victim is rich.

Part A: Write the above facts and rules in your Prolog program. Use the following names for the people: **`colMustard`**, **`profPlum`**, **`missScarlet`**, **`msGreen`**, **`mrBoddy`**. Be careful about how you encode (or don't encode) symmetric relationships like marriage - you don't want infinite loops! **`married(X,Y) :- married(Y,X) % INFINITE LOOP`**

Part B: Write a predicate, **`suspect/2`**, that determines who the suspects may be, i.e. who had a motive, given a victim.

```
?- suspect(Killer,mrBoddy)
Killer = suspect_name_1
Killer = suspect_name_2
etc.
```

Part C: Add a *single fact* to your database that will result in there being a unique suspect. Clearly indicate this line in your source comments so that it can be removed/added for grading.

```
?- suspect(Killer,mrBoddy)
Killer = unique_suspect.
```