Programming Project #2
Prolog Programming Assignment

Define and test the Prolog predicates described below. Each of your predicates ***must*** have the same name and signature as the examples in each of the ten problems. Your predicates must behave properly on all instances of valid input types.

Your submission should consist of a single source code text file that includes all facts, predicate definitions, and propositions.

Your file should be named `<your_net_id>.prolog`

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

- SWI-Prolog manual
- SWI-Prolog documentation
- Learn Prolog Now!

The Parameter Mode Indicator

Predicate signatures in Prolog are different from function signatures in C++ or Java. A C++ function signature will indicate the data types of a return value as well as the data types of any local function variables referenced in the association function body. For example, `int myFunction(float f, string s)`.

By contrast, a ***parameter mode indicator*** in Prolog gives information about the intended direction in which information carried by a predicate parameter is supposed to flow. Parameter mode indicators are are meta-symbols and not a formal part of the Prolog language but help in explaining intended semantics to the programmer. They are not used in source code itself.

There is no widely accepted agreement on parameter mode indicators in the Prolog community. A list of these symbols adopted by SWI-Prolog can be found in the Reference Manual in Section 4.1 here.

Note: The SWI-Prolog Reference Manual refers to parameters as “arguments”. Technically, predicates have parameters; functions and methods have arguments. I will not deduct points for any homework or exam responses if you call parameters “arguments”.

1) Divisible by both X and Y [10 points]

Description:

A user should be able to enter the predicate with three integer parameters. If the given first parameter is evenly divisible by both the second and third parameters, the predicate will evaluate to true, otherwise false.

Predicate Signature with Parameter Modes:

```
div-by-xy(+Integer, +Integer, +Integer)
```

Examples:

```
?- div-by-xy(24,3,4).
true.

?- div-by-xy(35,5,7).
true.

?- div-by-xy(50,5,3).
false.

?- div-by-xy(63,-7,3).
true.
```

2) List Product

Description:

Define a predicate `list_prod/2` that takes a list of numbers as a first parameter and determines the product of all of the list elements as output in the second parameter. The product of an empty list should be zero.

Predicate Signature with Parameter Modes:

```
list_prod(+List, −Number)
```

Examples:

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

?- list_prod([7,8,0,13], Product).
Product = 0.

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

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

3) Palindrome

Description:

Define a predicate `palindrome/1` that takes a list of any type of literal values or symbols as a single parameter and evaluates whether the list is the same both backward and forward, i.e. a "palindrome" list.

Note: The list to be evaluated may be heterogenous data types.

Predicate Signature with Parameter Modes:

```
palindrome(+List)
```

Examples:

```
?- palindrome([4,3,4]).
true.

?- palindrome([7,2,5,7]).
false.

?- palindrome([“hi”,4,i,4,"hi"]).
true.

?- palindrome([]).
true.

?- palindrome([a]).
true.
```

4) Second Minimum [10 points]

Description:

Define a predicate `secondMin/2` with the signature `secondMin(+List, -Min2)` where `Min2` is the second lowest *unique* valued element in some list of numbers, `List`. If the list has fewer than two unique elements, then your predicate should display the following, "`ERROR: List has fewer than two unique elements.`"

If one more elements of `List` is not a number, then your predicate should use `writeln(+String)` to display the following message for the first encounter of a non-number element, "`ERROR: "element" is not a number.`", where *`element`* is the value of the non-number element.

Your defintion may *not* use the built-in `sort/2` predicate as a helper predicate. However, you may define your own `mySort/2`.

Predicate Signature with Parameter Modes:

```
secondMin(+List, -Min2)
```

Examples:

```
?- secondMin([17,29,11,62,37,53], M2).
M2 = 17

?- secondMin([512], M2).
ERROR: List has fewer than two unique elements.

?- secondMin([7,5.2,3,6,-3.6,9,-2], M2).
M2 = -2

?- secondMin([12,2,b,7], M2).
ERROR: "b" is not a number.

?- secondMin([3,3,3], M2).
ERROR: List has fewer than two unique elements.
```

5) Classify

Description:

Define a predicate `classify/3` that takes a list of integers as an parameter and generates two lists, the first containing containing the even numbers from the original list and the second sublist containing the odd numbers from the original list.

Predicate Signature with Parameter Modes:

```
classify(+List, -Even, -Odd)
```

Examples:

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

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

?- classify([-4,11,-7,9,0], Even, Odd).
Even = [-4,0]
Odd = [11,-7,9]

?- classify([5,13,29], Even, Odd).
Even = []
Odd = [5,13,29]

?- classify([], Even, Odd).
Even = []
Odd = []
```

6) Bookends

Description:

Design a predicate `bookends/3` whose three parameters are all lists. The predicate verifies if parameter 1 is a prefix of parameter 3, ***and*** if parameter 2 is a suffix of parameter 3.

Note: The end of the list in parameter 1 may overlap with the beginning of the list in parameter 2.

Predicate Signature with Parameter Modes:

```
bookends(+List1, +List2, +List3)
```

Examples:

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

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

?- bookends([8,7,3],[3,4],[8,7,3,4]).
true.

?- bookends([6],[9,3],[6,9,3,7]).
false.

?- bookends([],[],[2,4,6]).
true.

?- bookends([23],[23],[23]).
true.
```

7) Subslice

Description:

Design a predicate `subslice/2` that tests if the list in parameter 1 is a contiguous series of elements anywhere within in the list in parameter 2.

Predicate Signature with Parameter Modes:

```
subslice(+List1, +List2)
```

Examples:

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

?- subslice([8,13],[3,4,8,13,7]).
true.

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

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

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

8) Shift

Description:

Design a predicate `shift/3` that "shifts" or "rotates" a list *N* places to the left. *N* may be a negative number, i.e. rotate to the right. Note that the rotated list should be the same length as the original list.

Predicate Signature with Parameter Modes:

```
shift(+List, +Integer, -List)
```

Examples:

```
?- shift([a,b,c,d,e,f,g,h],3,Shifted).
Shifted = [d,e,f,g,h,a,b,c]

?- shift([1,2,3,4,5],1,Shifted).
Shifted = [2,3,4,5,1]

?- shift([a,b,c,d,e,f,g,h],-2,Shifted).
Shifted = [g,h,a,b,c,d,e,f]
```

9) Luhn Algorithm

Description:

Design a predicate `luhn/1` that is an implementation of the Luhn Algorithm and returns `true` if the parameter is an integer that passes the Luhn test and `false` otherwise.

Refer to these resources for a description of the Luhn Algorithm:

- Rosetta Code (Luhn Test of Credit Card Numbers) [link]
- Wikipedia (Luhn Algorithm) [link]

Predicate Signature with Parameter Modes:

`luhn(+Integer)`

Examples:

```
?- luhn(799273987104).
true.

?- luhn(49927398717).
false.

?- luhn(49927398716).
true.
```

10) Graph

Description:

Design *two* predicates `path/2` and `cycle/1` that determine structures within a graph whose directed edges are encoded with given 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.

Note: Your solution should avoid infinite recursion.

Note: The Knowledge Base of edges in the example below is for explanation only. You are just responsible for the definitions of `path/2` and `cycle/1`. The Knowledge Base edges used for grading will be different.

Predicate Signature with Parameter Modes:

```
path(+Node1, +Node2)
cycle(+Node)
```

Examples:

```
% 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.
```