Programming Project #2
Prolog Programming Assignment

Create definitions for the nine Prolog predicates described on the following pages. Each of your predicate definitions ***must*** have the same name and signature as the examples in each of the nine problems. Your predicates must behave properly on all instances of valid input types, not just the examples provided here.

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`, for example `cid021000.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 method signatures in 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 the format of 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***; while functions and methods have arguments. I will not deduct points for any homework or exam responses if you call parameters “arguments”.

1) List Product [10 points]

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 is a special case and for the purposes of this assignment is defined as `0` (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.
```

2) Is Prime [10 points]

Description:

Define a predicate `is_prime/1` that takes an integer as its single parameter and evaluates to to `true` if the integer is prime and `false` otherwise.

Predicate Signature with Parameter Modes:

```
is_prime(+Integer)
```

Examples:

```
?- is_prime(31).
true.

?- is_prime(24).
false.

?- is_prime(4096).
false.

?- is_prime(1013).
true.
```

3) 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 any built-in sort predicates from the standard library as a helper predicate, such as `sort/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.
```

4) Classify [10 points]

Description:

Define a predicate `classify/3` that takes a predicate as its first parameter, a list of elements as the second parameter and two generated lists as the third and fourth parameters.

Predicate Signature with Parameter Modes:

```
classify(+Pred, +List, -List1, -List2)
```

Examples:

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

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

?- classify(integer, [4.2,11,7,9.7,0], Int, NotInt).
Int = [11,7,0]
NotInt = [4.2,9.7]

?- classify(string, [5, "seven",29], String, NotString).
String = ["hi"]
NotString = [5,29]

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

5) Bookends [10 points]

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 (Prefix) may overlap with the beginning of the list in parameter 2 (Suffix).

Predicate Signature with Parameter Modes:

```
bookends(+Prefix, +Suffix, +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.
```

6) Subslice [10 points]

Description:

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

Predicate Signature with Parameter Modes:

```
subslice(+Sub, +List)
```

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.
```

7) Rotate [10 points]

Description:

Design a predicate `shift/3` that "rotates" or "shifts" a list *N* places to the left. Elements that are rotated off past the beginning or end of a list are appended to the opposite end. *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:

```
rotate(+InputList, +Integer, -RotatedList)
```

Examples:

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

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

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

8) Luhn Algorithm [10 points]

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.
```

9) Zebra Puzzle [20 points]

Description:

Design a predicate that solves the following "Zebra Puzzle" https://en.wikipedia.org/wiki/Zebra_Puzzle.

Step into the vibrant atmosphere of the Riffs and Mystique Festival, where five unique bands from across Europe are gearing up to play their hearts out. Your challenge is to crack this Zebra Puzzle by matching each band's genre, native country, clothing color, the year they started, and the size of their social media following.

- *Clothing*: black, blue, green, orange, yellow
- *Country*: Croatia, Denmark, Ireland, Slovakia, Austria
- *Genre*: gothic, hard rock, heavy metal, progressive rock, psychodelic rock
- *Followers*: 50k, 150k, 200k, 250k, 300k
- *Year*: 2000, 2005, 2008, 2009, 2015

Determine (1) which band wears black, and (2) which band hails from Denmark

Examples:

Implement your solution by defining the predicates `black/1` and `denmark/1`.

```
?- black(SomeVar).
SomeVar = 4 % if band #4 were the answer

?- denmark(B).
B = 2 % if band #2 were the answer
```

Multiple strategies to solve Zebra puzzles in Prolog can be found here:

- https://rosettacode.org/wiki/Zebra_puzzle#Prolog

Note: It is not required to display all values for all bands (like the Rosetta Code examples) but it may help your debugging.

Sixteen Facts:

1. The band that started in 2009 is in the fourth position.
2. The band with 50,000 followers is *somewhere* to the right of the band wearing Black clothing.
3. The band with 150,000 followers is *next to* the band from Austria.
4. The band with 300,000 followers is located *somewhere between* the band that started in 1980 and the band with 250,000 followers, in that order.
5. The band wearing Black clothing is *somewhere* between the bands that started in 2008 and 2000, in that order.
6. The band that started in 2000 is next to the band wearing Green clothing.
7. The band from Austria is immediately before the band from Denmark.
8. The band wearing Black clothing is somewhere to the left of the Heavy Metal band.
9. The Progressive Rock band is in the first position.
10. The band with 50,000 followers is *somewhere* between the band wearing Yellow clothing and the band with 300,000 followers, in that order.
11. The band from Slovakia is *immediately before* the band from Ireland.
12. The band that started in 2000 is *somewhere between* the bands that started in 2015 and 2009, in that order.
13. The Gothic band is immediately after the band from Slovakia.
14. The band that started in 2009 is somewhere between the band wearing Yellow clothing and the band that started in 2005, in that order.
15. The band from Croatia is *somewhere to the left* of the band wearing Blue clothing.
16. The band wearing Blue clothing is *somewhere to the left* of the Psychedelic band.