Name: NIM:

Homework 4
TK2ICM: *Logic Programming* (CSH4Y3)
Second Term 2018-2019

Due date : Wednesday, April 17, 2019 at 05:00 a.m. CeLoE time
Type : ***open all***, *individual, cooperation is allowed*

Instruction:

1. This homework is due **Wednesday April 17 at 05:00 a.m. CeLoE time**. Please submit your homework through the submission slot at CeLoE. You are allowed to discuss these problems with other class participants, but make sure that you solve the problems individually. Copying answers from elsewhere without understanding them will not enhance your knowledge.
2. You may use any reference (books, slides, internet) as well as ask other students who are not enrolled to this class.
3. Discussion with fellow participants is encourage as long as you do not copy or rewrite the codes without understanding them. Try solving the problem individually before consulting other students.
4. Try to solve all problem in a time constraint to make you accustomed to the exam condition.
5. Use the predicate name as described in each of the problem. **The name of the predicate must be precisely identical**. Typographical error may lead to the cancellation of your points.
6. Submit your work to the provided slot at CeLoE under the file name `Hw4-<your_name>.pl`. For example: `Hw4-Albert.pl`. Please see an information regarding your nickname at Google Classroom.

1 Arithmetic and Geometric Mean

Remark 1 This problem is worth 20 points.

Suppose we have n numbers $a_1, a_2, \cdots, a_n$, the *arithmetic mean* of these numbers, denoted by A, is defined by the formula

$$A = \frac{1}{n}\sum_{i=1}^{n} a_i = \frac{a_1 + a_2 + \cdots + a_n}{n}. \tag{1}$$

For example, the arithmetic mean of 1, 2, 2, and 8 is $\frac{1}{4}(1+2+2+8) = \frac{13}{4} = 3.250$. The *geometric mean* of n numbers $a_1, a_2, \ldots, a_n$, denoted by G, is defined by the formula

$$G = \sqrt[n]{\prod_{i=1}^{n} a_i} = \sqrt[n]{a_1 \cdot a_2 \cdot \cdots \cdot a_n}. \tag{2}$$

For example, the geometric mean of 1, 2, 2, and 8 is $\sqrt[4]{1 \cdot 2 \cdot 2 \cdot 8} = \sqrt[4]{32} = 2.378$.

In this problem, your task is to write two predicates, namely `arithmean/2` and `geomean/2`, that respectively calculate the arithmetic and geometric means of a list of numbers. *The numbers are assumed to be nonnegative.* The length of the list is not specified, but the list is assumed to be non-empty. Several test case examples are:

- `?- arithmean([1,2,2,8],A).` returns `A = 3.25`.
- `?- geomean([1,2,2,8],G).` returns `G = 2.378414230005442`.
- `?- arithmean([4,2,4,8],A).` returns `A = 4.5`.
- `?- geomean([4,2,4,8],G).` returns `G = 4.0`.
- `?- arithmean([0,2,3,5,7,11],A).` returns `A = 4.666666666666667`.
- `?- geomean([0,2,3,5,7,11],G).` returns `G = 0.0`.
- `?- arithmean([8,8],A).` returns `A = 8`.
- `?- geomean([8,8],A).` returns `A = 8.0`.

Note: the first argument must be instantiated.

2 $\gcd$ and lcm of Multiple Numbers

Remark 2 This problem is worth 20 points.

Given n nonnegative integers $a_1, a_2, \ldots, a_n$, not all of them are zero, the greatest common divisor of $a_1, a_2, \ldots, a_n$ is the largest integer d that divides each a_i for all $1 \leq i \leq n$. The number d is denoted by $\gcd(a_1, a_2, \ldots, a_n)$. In elementary school, we learnt about finding the $\gcd$ of multiple numbers using prime factorization. However, this method is inefficient. In Homework 3 (*midterm exam preparation*), we've discussed the implementation of Euclid's algorithm for finding the $\gcd$ of two numbers in Prolog. We can combine Euclid's algorithm and the following theorem to calculate the $\gcd$ of multiple numbers.

Theorem 1 If a, b, and c are three numbers and not all of them are zero, then

$$\gcd(\gcd(a,b),c) = \gcd(a,\gcd(b,c)) = \gcd(\gcd(a,c),\gcd b), \tag{3}$$

thus, the $\gcd$ of three numbers a, b, and c can be denoted by $\gcd(a,b,c)$.

As a result, we also have the following corollary.

Corollary 1 If $a_1, a_2, \ldots, a_n$ are nonnegative integers and not all of them are zero, then

$$\begin{aligned} &\gcd(a_1, a_2, \ldots, a_n) = \gcd(a_1, \gcd(a_2, a_3 \ldots, a_n)), \text{ where} \\ &\gcd(a_2, a_3, \ldots, a_n) = \gcd(a_2, \gcd(a_3, a_4, \ldots, a_n)), \text{ and so on.} \end{aligned} \tag{4}$$

For instance, we can compute $\gcd(48, 60, 72, 84)$ as follows:

$$\begin{aligned} \gcd(48, 60, 72, 84) &= \gcd(48, \gcd(60, 72, 84)) \\ \gcd(60, 72, 84) &= \gcd(60, \gcd(72, 84)) \\ \gcd(72, 84) &= 12, \end{aligned}$$

hence

$$\begin{aligned} \gcd(60, 72, 84) &= \gcd(60, 12) = 12 \\ \gcd(48, \gcd(60, 72, 84)) &= \gcd(48, 12) = 12. \end{aligned}$$

Thus, we have $\gcd(48, 60, 72, 84) = 12$.

In elementary school, we've also learnt about the least common multiple of n positive integers. Given n positive integers $a_1, a_2 \ldots, a_n$, the least common multiple of these numbers (denoted by $\mathrm{lcm}(a_1, a_2, \ldots, a_n)$) is the smallest multiple of all a_i where $1 \leq i \leq n$. Finding lcm of two numbers can done efficiently using the following theorem.

Theorem 2 If a and b are two positive integers, we have $a \cdot b = \gcd(a,b) \cdot \mathrm{lcm}(a,b)$.

For example, we have $\mathrm{lcm}(8,12) = 24$, this value can be computed using the formula $\mathrm{lcm}(8,12) = (8 \cdot 12) \,/\, \gcd(8,12) = 24$. The lcm of three numbers can be calculated using the following theorem.

Theorem 3 If a, b, and c are three positive integers, then

$$\mathrm{lcm}(\mathrm{lcm}(a,b),c) = \mathrm{lcm}(a,\mathrm{lcm}(b,c)) = \mathrm{lcm}(\mathrm{lcm}(a,c),\mathrm{lcm}\, b), \tag{5}$$

thus, the lcm of three numbers a, b, and c can be denoted by $\mathrm{lcm}(a,b,c)$.

As a result, we have the following corollary.

Corollary 2 If $a_1, a_2, \ldots, a_n$ are positive integers, then

$$\begin{aligned} &\mathrm{lcm}\,(a_1, a_2, \ldots, a_n) = \mathrm{lcm}\,(a_1, \mathrm{lcm}\,(a_2, a_3 \ldots, a_n))\text{, where} \\ &\mathrm{lcm}\,(a_2, a_3, \ldots, a_n) = \mathrm{lcm}\,(a_2, \mathrm{lcm}\,(a_3, a_4, \ldots, a_n))\text{, and so on.} \end{aligned} \tag{6}$$

In this problem, your task is to define the predicates `gcd/2` and `lcm/2` that respectively calculate the gcd and lcm of multiple numbers in a particular list. You may use Euclid's algorithm in Homework 3. The length of the list is not specified, but the list is assumed to be non-empty. In addition, **the elements of the list are assumed to be positive**. Several test case examples are:

- `gcd([48,60,72,84],X)` returns `X = 12`.
- `lcm([48,60,72,84],X)` returns `X = 5040`.
- `gcd([720,900,1080],X)` returns `X = 180`.
- `lcm([720,900,1080],X)` returns `X = 10800`.
- `gcd([2,3,5,7,11],X)` returns `X = 1`.
- `lcm([2,3,5,7,11],X)` returns `X = 2310`.
- `gcd([31,62],X)` returns `X = 31`.
- `lcm([31,62],X)` returns `X = 62`.
- `gcd([9],X)` returns `X = 9`.
- `lcm([9],X)` returns `X = 9`.

Note: the first argument must be instantiated.

3 Prime Numbers

Remark 3 This problem is worth 20 points.

Prime numbers are important objects in cryptography. Mostly, cryptographer uses a positive prime numbers for their security parameters. A positive integer $n > 1$ is prime if it has exactly two positive divisors, namely 1 and n itself. A positive number that is not prime is called composite. Primality testing is a procedure to test whether a particular integer n is a prime or composite. We can perform a straightforward naive primality testing using this definition, i.e., we compute the remainders of n when it is divided by any integer $i \in [2, n-1]$. If any of this remainder is equal to 0, then we conclude that n is composite. However, we can reduce the number of iteration by considering the following theorem.

Theorem 4 Suppose n is a composite number, then n has a prime factor which is less than or equal to $\lfloor \sqrt{n} \rfloor$.

Using the above theorem, we can construct a procedure to test whether a positive integer is prime. An example of procedural program for primality testing is given in Python as follows:

```
def IsPrime(n):
    import math
    N = math.floor(math.sqrt(n))
    prime = True; i = 3
    if n == 1:
        prime = False
    if n == 2:
        prime = True
    if n > 2 and n%2 == 0:
        prime = False
    while (prime == True) and (i <= N):
        if (n%i == 0):
            prime = False
        else:
            i += 2
    return prime
```

Your tasks are as follows:

(a). **[10 points]** Write a predicate `isPrime/1` that takes a positive integer as argument and returns **true** if that integer is prime and **false** otherwise. Several test case examples:

- `?- isPrime(2).` returns **true**.
- `?- isPrime(3).` returns **true**.
- `?- isPrime(10).` returns **false**.
- `?- isPrime(1997).` returns **true**.
- `?- isPrime(2017).` returns **true**.
- `?- isPrime(2019).` returns **false**.
- `?- isPrime(17081949).` returns **false**.
- `?- isPrime(17081951).` returns **true**.
- `?- isPrime(99999989).` returns **true**.

- `?- isPrime(999999899).` returns **false**.
- `?- isPrime(1000000009).` returns **true**.

(b). **[10 points]** Write a predicate `prime_list/3` whose first and second arguments are positive integers and the third argument is a list of primes between the first and the second arguments (inclusive). If the first argument is larger than the second one, the values of the first and the second arguments are reversed. Several test case examples:

- `?- prime_list(71,95,L).` returns `L = [71, 73, 79, 83, 89]`.
- `?- prime_list(101,80,L).` returns `L = [83, 89, 97, 101]`.
- `?- prime_list(1997,1997,L).` returns `L = [1997]`.
- `?- prime_list(1202,1212,L).` returns `L = []`.
- `?- prime_list(17081945,17082018,L).` returns `L = [17081947, 17081951, 17081959, 17081993, 17082001, 17082017]`.
- `?- prime_list(1000000000,1000000090,L).` returns `L = [1000000007, 1000000009, 1000000021, 1000000033, 1000000087]`.

Note: the first and the second arguments must be instantiated.

4 Splitting a List

Remark 4 This problem is worth 20 points.

Define a predicate `split/4` that takes four arguments, `split(L,X,P1,P2)` splits the list `L` of numbers into two lists: one containing the numbers greater than or equal to `X` (the list `P1`), and the other containing the numbers less than `X` (the list `P2`). Several test case examples:

- `?- split([3,4,-5,-1,0,4,-9],1,P1,P2).` returns

  ```
  P1 = [3,4,4],
  P2 = [-5,-1,0,-9].
  ```

- `?- split([3,4,-5,-1,0,4,-9],-9,P1,P2).` returns

  ```
  P1 = [3,4,-5,-1,0,4,-9],
  P2 = [].
  ```

- `?- split([3,4,-5,-1,0,4,-9],9,P1,P2).` returns

  ```
  P1 = [],
  P2 = [3,4,-5,-1,0,4,-9].
  ```

- `?- split([2,1,3,4,0,5],3,P1,P2).` returns

  ```
  P1 = [3,4,5],
  P2 = [2,1,0].
  ```

Note: the first and the second arguments must be instantiated.

5 Logic Puzzle: The Three Houses

Remark 5 This problem is adapted from Exercise 6.4 in the textbook *Learn Prolog Now!* and it is worth 20 points.

There is a street with three neighboring houses that all have a different color. They are red, blue, and green. People of different nationalities live in the different houses and they all have a different pet. Here are some more facts about them:

- The Englishman lives in the red house.
- The jaguar is the pet of the Spanish family.
- The Japanese lives to the right of the snail keeper.
- The snail keeper lives to the left of the blue house.

We are interested to solve the following questions:

(a). *Who keeps the zebra?*

(b). *What color is the Japanese house?*

To solve these problems, you can think of a representation for the houses and the street. Then, you can code the four constraints in Prolog. Here are some useful rules for solving this problems:

- First, we can represent the street as a list, `Street = [H1,H2,H3]`, where `H1`, `H2`, and `H3` represent the houses. *Ignore the singleton variable notification in the interpreter*.
- The immediate left and right neighbors can be defined as follows:

```
neighbor(L,R,[L,R|_]).
neighbor(L,R,[_|T]):- neighbor (L,R,T).
% two rules above state that house L is to the left of house R
```

Your task are ask follows:

(a). **[10 points]** Complete the following rule `owns_zebra_nationality(X)` that returns the nationality of the person who keeps the zebra.

```
owns_zebra_nationality(X) :-
    Street = [H1,H2,H3], % the street is the list of the three houses
    % the first constrain,
    % the second constrain,
    % the third constrain,
    % the fourth constrain,
    % the clause to find the zebra owner.
```

(b). **[10 points]** Complete the following rule `house_color_japanese(X)` that returns the color of the Japanese's house.

```
house_color_japanese(X) :-
    Street = [H1,H2,H3], % the street is the list of the three houses
```

```
% the first constrain,
% the second constrain,
% the third constrain,
% the fourth constrain,
% the clause to find the color of the japanese house.
```