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

Day, date	:	Tuesday, March 26, 2019
Duration	:	**60 minutes**
Type	:	***open all***, *individual (no cooperation between/among class participants)*

Instruction:

1. You are not allowed to discuss these problems with other class participants.
2. You may use any reference (books, slides, internet) as well as other students who are not enrolled to this class.
3. 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.
4. Submit your work to the provided slot at CeLoE under the file name `PS4-<your_name>.pl`. For example: `PS4-Albert.pl`. Please see an information regarding your nickname at google classroom.

1 Adding with 1

Problem 1 (20 points) Write a predicate `addone/2` whose first argument is a list of integers, and whose second argument is the list of integers obtained by adding 1 to each integer in the first list. Examples:

- `?- addone([1,2,7,2],X).` returns `X = [2,3,8,3]`.
- `?- addone(X,[-1,-4,2,5]).` returns `X = [-2, -5, 1, 4]`.

(Note: at least one of the arguments in `addone/2` must be instantiated.

2 List with Identical Element(s)

Problem 2 (20 points) Write a predicate `sameElemList/1` that takes a list as an argument and it returns true whenever the list has identical elements (i.e., the elements of the list are all the same). Examples:

- `?- sameElemList([a,a,a,a,a]).` returns **true**.
- `?- sameElemList([a,a,1,a,a]).` returns **false**.

3 Scalar Multiplication

Problem 3 (20 points) In mathematics, an n-dimensional vector is a list of numbers of length n. For example, `[2,5,12]` is a 3-dimensional vector, and `[45,27,3,-4,6]` is a 5-dimensional vector. One of the basic operations on vectors is scalar multiplication. In this operation, every element of a vector is multiplied by some number. For example, if we scalar multiply the 3-dimensional vector `[2,7,4]` by 3 the result is the 3-dimensional vector `[6,21,12]`. Write a ternary predicate `scalarMult` whose first argument is an number, its second argument is a list of numbers, and its third argument is the result of scalar multiplying the second argument by the first. For example, we have:

- `?- scalarMult(3,[2,7,4],Result).` returns `Result = [6,21,12]`.
- `?- scalarMult(5,[2,7],Result).` returns `Result = [10,35]`.
- `?- scalarMult(5,[3,2,-1],Result).` returns `Result = [15,10,-5]`.
- `?- scalarMult(3,Result,[6,21,12]).` returns `Result = [2,7,4]`.
- `?- scalarMult(5,Result,[10,35]).` returns `Result = [2,7]`.
- `?- scalarMult(Result,[2,7],[10,35]).` returns `Result = 5`.

4 Vectors' Addition

Problem 4 (20 points) In linear algebra, two vectors of the same dimension can be added. The result of this addition is a vector of the same size whose i-th element is the sum of the i-th elements of the first and the second vectors. For instance, the addition of `[2,5,6]` and `[3,4,1]` is `[5,9,7]`. Write a ternary predicate `add` whose first and second argument are list of numbers of the same size, and the third argument is the (vector) addition of the first and second argument. If the dimension do not match, write "`dimension error`" instead. For example, we have:

- `?- add([1,2,3],[4,5,6],X).` returns `X = [5,7,9]`.
- `?- add([-1,0,1],[2,3,-3],X).` returns `X = [1,3,-2]`.
- `?- add([-1,0,1],X,[1,3,-2]).` returns `X = [2,3,-3]`.
- `?- add(X,[2,3,-3],[1,3,-2]).` returns `X = [-1,0,1]`.
- `?- add([-1,0],[2,3,-3],X).` returns `X = "dimension error"`.
- `?- add([-1,0,1],[3,-3],X).` returns `X = "dimension error"`.

5 Dot Product

Problem 5 (20 points) Another fundamental operation on vectors is the dot product. This operation combines two vectors of the same dimension and yields a number as a result. The operation is carried out as follows: the corresponding elements of the two vectors are multiplied, and the results added. For example, the dot product of `[2,5,6]` and `[3,4,1]` is $6 + 20 + 6$, that is, 32. Write a ternary predicate `dot` whose first argument is a list of numbers, whose second argument is a list of numbers of the same length as the first, and whose third argument is the dot product of the first argument with the second. If the dimensions do not match, write "`dimension error`" instead. For example, we have:

- `?- dot([2,5,6],[3,4,1],Result).` returns `Result = 32`.
- `?- dot([1,0,-1],[-1,2,2],Result).` returns `Result = -3`.
- `?- dot([1,0,-1],[-1,2],Result).` returns `Result = "dimension error"`.
- `?- dot([1,0],[-1,2,1],Result).` returns `Result = "dimension error"`.