CMPSC 460 – Principles of Programming Languages
Homework #1
Spring 2017

Total Points: 100

Due Date: Thursday February 16th, 2017

Question 1: (20 points)

Write a recursive function called `cross-product` that takes two lists of numbers and returns a list of sub-lists where each sub-list represents the Cartesian product of one element from `list1` with the elements of `list2`.

```
> (cross-product '(2 3) '(1 2 3 4 5))
```

'((2 4 6 8 10) (3 6 9 12 15))

Question 2: (80 points)

Use Scheme to write an interpreter for a small language. This language will deal with arithmetic expressions that can be directly applied to arrays of numbers. Your code should be limited to the Scheme features and instruction that we discussed in class. Other features, such as set!, that are not discussed in class are not allowed.

The interpreter should accept as input the program as an s-expression, and should produce as an output the result of the program. The *read* function will help you convert the input from a string of characters to a Scheme list. For the new language, an s-expression has the following format:

- `array`
- `(s-expr operator)`
- `(s-expr operator s-expr)`

A program can either be an s-expression or an array. The value of the array is the array itself. On the other hand, the value of an s-expression is either an array or a number. Any of the allowed operators will expect an array as an input. You interpreter should issue an error when an atom or an array that is not only numbers is passed to an operator. In addition, we also assume that the elements of an array will always be one or more numbers.

An array is a list of one or more numbers that are enclosed inside square brackets. The list of numbers is space separated. For example, [4 -6 10 5] is an array of four integer numbers.
There are two main types of operators:

- ***Unary operators***: `add, average, absum, max, min, minmax, multiply, subtract, x2, x3`
- ***Binary operators***: `plus, times, minus`

Expression	***Output***
`array`	The array itself
`(array add)`	All the elements of the array are summed together
`(array absum)`	The absolute of all the elements of the array are added to each other
`(array multiply)`	All the elements of the array are multiplied
`(array subtract)`	All the elements of the array are subtracted from each other
`(array average)`	The average of all the elements of the array
`(array min)`	Returns the smallest number in *array*
`(array max)`	Returns the largest number in *array*
`(array minmax)`	Returns a list that contains the largest and smallest numbers in *array*
`(array x2)`	Returns a list whose elements are double the elements of *array*
`(array x3)`	Returns a list whose elements are triple the elements of *array*
`(array1 plus array2)`	Each element of *array1* is added to its respective element from *array2*
`(array1 minus array2)`	Each element of *array2* is subtracted from its respective element from *array1*
`(array1 times array2)`	Each element of *array1* is multiplied with its respective element from *array2*

Your interpreter should follow the Read Evaluate Print Loop (REPL) structure and should have the following features:

- All the *operators'* (except *add* and *plus*) implementation should use recursive functions and should not use: *eval*, *map*, or *apply*.
- For *add* and *plus* operators, your functions in this case should not be recursive; instead you will implement these features using Scheme's *eval*, *map* or *apply* functions.
- You can assume that the binary operators will have as arguments two equally sized arrays.

Some examples:

> 5
Illegal value for an s-expression.

> [4 0 10 5]
'(4 0 10 5)

> ([4 0 -10 5] x2)
'(8 0 -20 10)

> ([4 0 10 5] x3)
'(12 0 30 15)

> [4.2 2.3 10 5.5]
'(4.2 2.3 10 5.5)

> ([-40 60 100 55] max)
100

> ([40 60 100 55] minmax)
'(40 100)

> ([40 60 100 55] average)
63 ¾

> ([40 60 100 55] add)
255

> ([400 60 100 50] subtract)
190

> ([1 2 3 4] plus [40 60 100 55])
'(41 62 103 59)

> ([1 2 3 4] plus ([1 2 3 4] x3))
'(4 8 12 16)

> ([40 60 100 55] minus [1 2 3 4])
'(39 58 97 51)

What to submit:

Pleasesubmit the following:

- The Scheme programs for questions 1 & 2.
- Enough test cases and screenshots of the results to demonstrate that your program is working properly.