SCALA HOMEWORK DUE: OCTOBER 21, 11:59PM EST

1. Instructions

1.1. **What to look at.** The only references you may use for this lab are:

- The scala documentation: `https://docs.scala-lang.org/`
- Any material on canvas, including the scala cheetsheat and scala book that are linked to on the main canvas page.

In particular, do not use stackoverflow or chegg for two reasons: (1) it will result in an academic integrity violation and (2) you are likely to get a hilariously incorrect answer from them (don't be an anecdote).

1.2. **How to collaborate.**

- Write down (in your code) the names of people you collaborate with.
- You *cannot* look at code/pseudocode
- You *cannot* copy/email/transmit code. Your code must be typed by you without looking at other people's code.
- You can discuss algorithmic strategies (pictures are fine, pseudocode is not)

1.3. **Getting the code and submitting the code.** Run **updatestarter** to get the **scalahw** directory and files. Submit to gradescope **Make sure to only commit scala code, and no other files into your repository.**

1.4. **Compiling and running the code.** The scalahw directory will contain the files tester.scala and scalahw.scala. Your code should go inside the **HW** object in scalahw.scala and **Tester** object in tester.scala. Remember, since scala uses the Java virtual machine, all of the code in each file has to reside inside an object or a class.

To compile your code you can do either of the following:

- From the linux shell (in the directory containing the code): **sbt compile** compiles the code and **sbt run** runs the code.
- From the sbt shell (this is faster): **compile** compiles your code and **run** runs your code. Sometimes if you get strange errors, **clean** can help. It will remove temporary files to allow you to recompile from scratch.

The entry point to the code is already set to be tester.scala, and it will call the code inside the HW object in scalahw.scala.

Notes:

- Do NOT use a return statement (i.e., "return" should not appear in any of your functions). If you find this confusing, refer to the scala slides.
- If you are seeing error message that point to the tester.scala file, the most likely reason is that the code you wrote in scalahw.scala is not returning answers of the correct type.
- All of your functions should be in the HW object in scalahw.scala, except for your testing code, which should be in the object in tester.scala. Make sure all of your functions have testing code.
- Do not use **var** or mutable state

Submit to Gradescope

2. Questions

Question 0. Your files will not compile until every function needed in this homework has a function signature in the scalahw.scala file. This includes function name, arguments and their types and return type specification (see q1 in scalahw.scala for an example). The function should also return a value of the correct type (as in q1 in scalahw.scala). Once this is done for all functions, then you can start adding code to make the functions correct. Note: If you choose to use **reduce** instead of **foldLeft** in these problems, you have to

make sure the list/vector your are applying it to is not empty. Reduce will crash on an empty list (because it will not have an initializeer) but foldLeft will work (because you give foldLeft the initializer). So I always recommend foldLeft.

Question 1. Fill in the q1() function in scalahw.scala. Given a number n, it should only use the tabulate function to return a list containing the first n square numbers (i.e., q1(3) should return a list containing 1,4,9).

Question 2. Fill in the q2() function in scalahw.scala. Given an integer n, it should only use the tabulate function to return a vector containing the square roots of the first n numbers (i.e., q1(3) should return a vector containing 1.0, 1.4142135623730951, 1.7320508075688772). The scala math library contains mathematical functions that could be useful.

Question 3. Fill in the q3() function in scalahw.scala. Its input is a Seq of Doubles (you can use Seq when you want to allow the possible inputs to be a vector, list etc.) and should return the sum of the items in the input. You can only use reduce, fold, or foldLeft (i.e., no loops, recursion, or indexing).

Question 4. Fill in the q4() function in scalahw.scala. Its input is a Seq of Doubles and should return the product of the items in the input. You can only use reduce, fold, or foldLeft (i.e., no loops, recursion, or indexing).

Question 5. Fill in the q5() function in scalahw.scala. Its input is a Seq of Doubles and should return the sum of the logs of the items in the input (use the log function from the scala math library). You can only use reduce, fold, or foldLeft (i.e., no loops, recursion, or indexing, no map).

Question 6. Fill in the q6() function in scalahw.scala. Its input is a Seq of tuples of Doubles (e.g., List((1.0, 2.0), (3.0, 4.0), (5.0, 6.0))) and should return a tuple containing the sum of the first items and the sum of the second items of the input tuples. In other words, in the example provided, the output should be (9.0, 12.0) because 9.0=1.0+3.0+5.0 and 12.0=2.0+4.0+6.0. You can only use foldLeft and only use it once, meaning that both sums have to be computed in the same foldLeft expression, so choose the state variable appropriately (i.e., it is not limited to just int, double, etc. but can be a tuple, etc.).

Question 7. Fill in the q7() function in scalahw.scala. Its input is a Seq of tuples of Doubles (e.g., List((1.0, 2.0), (3.0, 4.0), (5.0, 6.0))) and should return a tuple containing the sum of the first items and the max of the second items of the input tuples. Double.PositiveInfinity or Double.NegativeInfinity might be useful. You can only use foldLeft and only use it once.

Question 8. Fill in the q8() function in scalahw.scala. Its input is a Int n. It should return a tuple containing the sum of the first n numbers (starting with 1) and the product of the first n numbers. You can use 1 tabulate (or **to** or **until**) and 1 foldLeft.

Question 9. Fill in the q9() function in scalahw.scala. Its input is a Seq of ints. It should return the sum of the squares of the even numbers in the input. You have to use 1 map, 1 filter, and 1 reduce (or foldLeft).

Question 10. In the HW object, fill in the $q10()$ function. Its input is a Seq of Doubles. Its output is supposed to be $\sum_{i=0}^{n-1} input(i) * (i+1)$, where n is the number of elements in the input sequence (note in Scala that indexing uses parentheses instead of square brackets). For example q10(List(2.0, 3.0, 4.0)) should return $2.0 * 1 + 3.0 * 2 + 4.0 * 3 = 20.0$. Use only foldLeft (no indexing of the input and no other list/vector operations) to solve this problem. If you are stuck, think about what state you need to maintain.

2.1. **Preparation for the Last Question.** Computers have trouble with decimal numbers. For example, in Python (or almost any other language), what do you get when you type:

```
(0.1 + 0.2) == 0.3
```

If you guessed False, then you are correct. Due to rounding error during arithmetic, the left hand side does not end up equalling the right hand side on a computer. When adding many floating point numbers together, the rounding error can accumulate significantly. Fortunately, there is a nice numerical algorithm for addition that controls the error when adding positive numbers (if you have positive and negative numbers, maintain a sum of the positive ones and a sum of the negative ones separately). This algorithm is called Neumaier Summation. Wikipedia gives its pseudocode as follows (`https://en.wikipedia.org/wiki/Kahan_summation_algorithm`):

```
function NeumaierSum(input)
    var sum = 0.0
    var c = 0.0                         // A running compensation for lost low-order bits.
    for i = 1 to input.length do
        var t = sum + input[i]
        if |sum| >= |input[i]| then
            c += (sum - t) + input[i] // If sum bigger, tracks low-order digits of input[i]
        else
            c += (input[i] - t) + sum // Else tracks low-order digits of sum.
        endif
        sum = t
    return sum + c                      // Correction only applied once in the very end.
```

It maintains a running sum and also a compensator c that keeps track of things lost to rounding error.

This pseudocode uses modifiable state (the vars).

Question 11. Modify the NeumaierSum pseudocode so that it does not use any modifiable state. In the HW object, create the function $q11()$ whose input is a Seq[Double] to do this summation. Use the **case class Neumaier** (defined at the top in scalahw.scala) to keep track of the state (sum and compensator c). Use foldLeft (no indexing) to implement the summation algorithm without any loops (if you are stuck, examine how we used foldLeft in class). The state that foldLeft uses should be a case class Neumaier (so you should know how to create this thing and how to pull information out of it). Remember Do not use vars!!!. Do not use the return keyword. You must update the function definition to specify its return type.