CMPT 858 HOME EXERCISE 4

STREAMS

Within this problem, you will explore non-strict computation by creating streams in Scala.

In pursuing this home exercise, as in the last one, you may wish to recall that given an existing Stream[T] stream and a value v of type T, one can append an element on to that stream with the following syntax:

v #:: stream

1. Define a function which returns a Stream[Long] whose n^{th} element (starting from 1) represents n! (i.e., n*(n-1)*(n-2)*...*1). Please note that you can decide what parameters this function takes, as long as it can return this specified stream.
2. Create a function (parameterized by type T) which, given two streams A and B of type T, returns a stream whose successive elements alternate are drawn from successive elements of A and B (in strict alternating succession between these two streams). That is, all elements of the returned stream with odd numbered indices (counting from 1) should represent the successive elements of A, and all elements of the returned stream with even indices should be drawn from B. In other words, the first element of the returned stream should be the first element of A, second element of the returned stream should be the first element of B, the third element of the returned stream should be the second element of A, the fourth element of the returned stream should be the fourth element of B, and so on.
3. See if you can create a function (again parameterized in whatever way you prefer) that can return a Stream of Longs, where each element of that returned stream is represents each successively larger prime number (represented as a long). (As with problem 1, such a stream would typically require that the function be called with the appropriate arguments). This problem is harder than the two above, and (as with all exercises) you are urged to try it, but to not spend hours on it.

Due in class October 11, 2016