built on 2017/11/22 at 10:25:13 ***due:*** *sat dec 2 @ 11:59pm*

This is a programming assignment. There's a starter package containing method stubs. When you're done, make a single zip file and upload it to Canvas.

- You can define as many helper functions as necessary. Be mindful of what you should expose to outside your function.
- You are going to be graded on style as well as on correctness.
- Test your code!

Task 1: Using Higher-Order Library Functions (20 points)

For this task, save your code in `UseLib.scala`

To dust off programming with collections, you will implement the following functions. Don't reinvent the wheel.

(1) Write a function `def onlyBeginsWithLower(xs: Vector[String]): Vector[String]` that takes a string vector and returns a string vector that has only the strings from the input that start with a lower-case letter. The empty string doesn't start with a lower-case letter. You should use the `filter` method.

(2) Write a function `def longestString(xs: Vector[String]): Option[String]` that takes a string vector and returns the longest string in the vector. If the vector is empty, return None. Use `maxBy`.

(3) Write a function `def longestLowercase(xs: Vector[String]): Option[String]` that takes a string list and returns the longest string in the vector that begins with a lower-case letter. If there are no such strings, return None.

Task 2: Currying (20 points)

For this task, save your code in `Currying.scala` In this problem, your code *must* follow the functional style. Implement the following:

(1) Write a function `firstAnswer` of type

```
(A => Option[B]) => List[A] => B
```

(notice the arguments are curried). The first argument should be applied to elements of the second argument in order until the first time it returns `Some(v)` for some `v` and then `v` is the result of the call to `firstAnswer`. If the first argument returns `None` for all list elements, then `firstAnswer` should raise the exception `NoAnswer`.

(2) Write a function `allAnswers` of type

```
(A => Option[List[B]]) => List[A] => Option[List[B]]
```

(notice the arguments are curried). The first argument should be applied to elements of the second argument.

- If it returns `None` for any element, then the result for `allAnswers` is `None`.
- Else, the calls to the first argument will have produced

  ```
  [Some(lst1), Some(lst2), ..., Some(lstn)]
  ```

  and the result of `allAnswers` is `Some(lst)`, where `lst` is `lst1, lst2, ..., lstn` concatenated together (in that order).

Task 3: Ham, Spam, Spaghetti (20 points)

For this task, save your code in `Spaghetti.scala`

This problem involves working with lazy evaluation and streams. Your implementation for this problem will go inside `object Spaghetti`.

(a) **(10 points)** The spaghetti sequence is the sequence of integers: 1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, ... The sequence starts with the number 1. To generate next element, we read off the digits of the previous member, counting the number of digits in groups of the same digit, like so: 1 is read "one 1" giving rise to 11. Then, 11 is read "two 1" giving rise to 21. Then, 21 is read "one 2 one 1", i.e., 1211. Next, 1211 is read "one 1 one 2 two 1", i.e., 111221.

You'll implement `def spaghetti: Stream[String]` that gives a stream of the spaghetti sequence where each element is represented as a string.

(b) **(10 points)** An n-bit ham code is a sequence of n-bit strings constructed according to certain rules. For example,

n = 1: $H(1)$ = "0", "1"
n = 2: $H(2)$ = "00", "01", "11", "10"
n = 3: $H(3)$ = "000", "001", "011", "010", "110","111", "101", "100"

Your task: find a pattern in the example and come up with the construction rules. Then, implement `def ham: Stream[String]` that gives a stream listing Ham codes for $n = 1, 2, 3, \ldots$. That is, the ham stream starting with "0", "1", "00", "01", "11", "10", "000", "001", "011", "010", "110","111", "101", "100", ...

(Hint: $H(n)$ can be defined recursively using $H(n-1)$ and operations involving prepending a digit to the string and reversing a sequence.)

Task 4: Mining A Hay Stack (20 points)

For this task, save your code in `Mining.scala`

Your implementation for this problem will go inside `object Mining`.

You're given a data file containing all flight records for one year. An example for year 2008 is available for download at

https://cs.muic.mahidol.ac.th/~ktangwon/2008.csv.zip

This file has been zip compressed, so you should uncompress it before use. The input file (e.g., `2008.csv`) is comma-separated and the first line contains the field headers. The fields are explained here http://stat-computing.org/dataexpo/2009/the-data.html.

You will write a function that will give us a glimpse into this rich dataset. Implement a function `def onTimeRank(fileName: String): Vector[(String, Double)]` that takes in a filename (e.g., 2008.csv) and returns a list of (airline code, ontime percentage), sorted from best to worst ontime percentage. A flight is ontime if the `ArrDelay` field is a negative number, 0, or not a number. The field `UniqueCarrier` provides the airline code. The ontime percentage is calculated as the percentage of ontime flights compared to all flights using that carrier code.

You must use Future/Promise and/or parallel collections to help speed this up (not slow it down).

On 2008.csv, your code must run in under 45 seconds. There will be a timing competition for (extra) points.