Assigmmnent2

Part1

Implementation

```
  /**
   * Construct a stream of Integers starting with <code>start</code> and with successive elements being
   * greater than their predecessors by <code>step</code>.
   *
   * @param start the value of the first element.
   * @param step  the difference between successive elements.
   * @return a <code>ListLike[X]</code> with an infinite number of element (whose values are <code>x</code>,
   *         <code>x+step</code>, etc.).
   */
/  def from(start: Int, step: Int): ListLike[Int] = ??? // TO BE IMPLEMENTED

  def from(start: Int, step: Int): ListLike[Int] = MyLazyList(start, () => {from(start+step,step)} )
```

Unit test

Part2

Q1:

A) Are the heads of lazylist and MyLazyList both lazy? If not explain the difference
   - yes. Both are strict on its head, but lazy on its tail.

B) The tail of both lazylist and MyLazyList are both lazy. Explain the mechanism used to make the tail lazy
   - Lazy means its evaluation is deferred, only when invoked or needed, yields the tail of this stream.

Q2: Explain what the following code actually does and why is it needed?

def tail = lazyTail()

- Define a new ListLike[X] collection to receive the resturn value of function lazyTail(), in other words to yields the tail of the stream as a new LikstLike[X] collection

Q3: List all of the recursive calls that you can find in *MyLazyList* (give line numbers).

- Line 39 `flatMap`
- Line66 filter
- Line79 zip
- Line91 take
- Line109 drop
- Line345 apply
- Line 369 `continually`
- Line386 from
- Line396 from

Q4: List all of the mutable variables and mutable collections that you can find in *MyLazyList* (give line numbers).

- **Immutable**
  - **`Seq[X]` line126**
  - `Monadic[Y]` line39
  - **`LazyListLike[X]` line291**
  - `Monadic[Y]` line39
  - `Option[Int]` line141

- **Mutable**
  - **`MyLazyList[X]` line16**
  - `Monadic[Y]` line39
  - `Option[Int]` line141
  - **`ListLike[X]` line291**
  - **`LazyListLike[X]` line291**

Q5: What is the purpose of the *zip* method?

- Zip two stream , if one of the stream is empty, then have a EmptyList; otherwise get a stream append step by step

Q6: Why is there no *length* (or *size*) method for *MyLazyList*?

- Cause it's lazy , would always return None when this MyLazyList is not invoked.