1. (a) what is the chief way by which *MyLazyList* differs from *LazyList* (the built-in Scala class that does the same thing). Don't mention the methods that *MyLazyList* does or doesn't implement--I want to know what is the *structural* difference.
   (b) Why do you think there is this difference?
2. Explain what the following code actually does and why is it needed?

```
def tail = lazyTail()
```

3. List all of the recursive calls that you can find in *MyLazyList* (give line numbers).
4. List all of the mutable variables and mutable collections that you can find in *MyLazyList* (give line numbers).
5. What is the purpose of the *zip* method?
6. Why is there no *length* (or *size*) method for *MyLazyList*?

1. a). In MyLazyList, we use case class to distinguish between empty lists and non-empty lists. In LazyList, we use the state class to distinguish between empty and non-empty lists.

   In MyLazyList, this method should finish its function, but in LazyList, this method always only distinguishes empty or non-empty, and then calls the "impl" method to perform the actual function.

   In MyLazyList the tail is defined as a function, tail = lazyTail(). In LazyList the tail is defined as tail: LazyList[A] = state.

   b). The structure of the class is completely different.

   In LazyList, we don't have any case classes, and each method should distinguish between null values and call the "impl" method. But MyLazyList does not have these structures.

   In the tail definition, MyLazyList is defined as a function, but in LazyList, the tail is defined as LazyList [A]. These are completely different.

2. Define a method which will evaluate the lazyTail each time when this method was invoked. We need to record or get the value of current lazyTail to use in some methods/test cases (like x.tail.head) .If we don't define this method, it will cause some errors.

3. Recursive calls:
   Line 131: case MyLazyList(h, f) => inner(rs :+ h, f())

Line 383 def continually[X](x: => X): ListLike[X] = MyLazyList(x, () => continually(x))

Line 408 def from(start: Int, step: Int): ListLike[Int] = MyLazyList(start, () => from(start + step, step))

4. There is no mutable variables or mutable collections, because mutable variables are leaded by key word "var". And we need to import "scala.collection.mutable" to use mutable collections. Otherwise all collections in Scala are by default immutable.

5. Zip method is used to merge a ListLike stream to current ListLike stream and result is a ListLike stream of a tuple of the corresponding elements from the two original streams. When a streams is empty, this method will terminate.

6. Theoretically, we could have infinte number of elements in the list. Lazy means that when we need it will be created. In other word, the size of LazyList is indefinite. we could have infinte number of elements in the list.