Homework 1: Introduction to Scala

Due: Thursday, Jan 28 2016 1AM

This assignment has several "finger exercises" that introduce you to functional programming in Scala.

1 Setup

Before you start programming, you need to complete a few preliminary steps.

1. Download and start the course virtual machine. You will need it for all the assignments in this class.
2. Using the command-line, create a directory for your assignment (e.g., the `hw1` directory). Within this directory, create the directories `src/main/scala` and `src/test/scala`. For example, you could use the following commands:

```
mkdir hw1
cd hw1
mkdir src
mkdir src/main
mkdir src/main/scala
mkdir src/test
mkdir src/test/scala
```

3. Using a text editor, create the file `src/main/scala/Lecture1.scala` with the following contents:

```
object Lecture1 {

  val oddNumbers = 1 :: 3 :: 5 :: Nil

}
```

4. Using a text editor, create the file `src/test/scala/TestSuite.scala` with the following contents:

```
import Lecture1._

class TestSuite extends org.scalatest.FunSuite {

  test("oddNumbers properly defined") {
    assert(oddNumbers == List(1, 3, 5))
  }

}
```

5. From the command-line, start `sbt` and run the test suite. You should see output that looks like this:

```
[info] Updating {file:/Users/arjun/Teaching/cmpsci220/hw/lists/template/}template...
[info] Resolving jline#jline;2.12.1 ...
[info] Done updating.
[info] Compiling 1 Scala source to /Users/arjun/Teaching/cmpsci220/hw/lists/template/
    target/scala-2.11/classes...
[info] Compiling 1 Scala source to /Users/arjun/Teaching/cmpsci220/hw/lists/template/
    target/scala-2.11/test-classes...
[info] TestSuite:
[info] - oddNumbers properly defined
[info] Run completed in 421 milliseconds.
[info] Total number of tests run: 1
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 1, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 7 s, completed Jan 17, 2016 11:52:54 AM
```

   There should be no errors or warnings printed.

2 Exercises

For this assignment, you'll be writing several list-processing functions. You must place these within the `Lecture1` object that you created above. You must write tests cases, within the `TestSuite` class that you created above.

1. Write a function called `sumDouble` that consumes a `List[Int]` and produces an `Int`. The produced value should be double the sum of the list of integers.

2. Write a function called `removeZeroes` that consumes a `List[Int]` and produces a `List[Int]`. The produced list should be the same as the input list, but with all zeroes removed. The function must not change the order of elements.

3. Write a function called `countEvens` that consumes a `List[Int]` and produces an `Int` that represents that number of even numbers in the input list.

4. Write a function called `removeAlternating` that consumes a `List[String]` and produces a `List[String]` that has every other element in the input list.

   The first element of the input list must be in the output list. For example:

   ```
   assert(removeAlternating(List("A", "B")) == List("A"))
   assert(removeAlternating(List("A", "B")) != List("B"))
   ```

   The function must not change the order of elments.

5. Write a function called `isAscending` that consumes a `List[Int]` and produces a `Boolean` that is `true` if the numbers in the input list are in ascending order. Note that the input may have repeated numbers.

6. Write a function called `addSub` that consumes a `List[Int]` and produces an `Int`. The function should add all the elements in even position and subtract all the elements in odd position.

   Note that the first element of a list is considered "zeroth" element, thus it is in even position. For example, `addSub(List(10, 20, 30, 40))` should be `10 - 20 + 30 - 40`.

7. Write a function called `alternate` that consumes *two* `List[Int]` arguments and produces a `List[Int]`. The elements of the resulting list should alternate between the elements of the arguments. You may assume that the two arguments have the same length.

   For example:

   ```
   assert(alternate(List(1, 3, 5), List(2, 4, 6)) == List(1, 2, 3, 4, 5, 6))
   ```

8. Write a function called `fromTo` that takes two `Int`s as arguments and produces a `List[Int]`. The value of `fromTo(x, y)` should be the list of consecutive integers that start from and include x, going up to and excluding y. You may assume that $x < y$.

   For example:

   ```
   assert(fromTo(9, 13) == List(9, 10, 11, 12))
   ```

9. Write the following function:

   ```
   def insertOrdered(n: Int, lst: List[Int]): List[Int]
   ```

   Assuming that `lst` is in ascending order, `insertOrdered` should produce a list that is the same as the input, but with n inserted such that the order is preserved. For example, `insertOrdered(5, List(1, 3, 7, 9))` should be `List(1, 3, 5, 7, 9)`.

   You should assume that `lst` is in ascending order. Your function may produce any result or even throw an exception if it is not.

10. Write the following function:

    ```
    def sort(lst: List[Int]): List[Int]
    ```

    The result should be the sorted input list.

3 Hand In

From the `sbt` console, run the command `submit`. The command will create a file called `submission.tar.gz` in your assignment directory. Submit this file using Moodle.

For example, if the command runs successfully, you will see output similar to this:

```
Created submission.tar.gz. Upload this file to Moodle.
[success] Total time: 0 s, completed Jan 17, 2016 12:55:55 PM
```

Note: The command will not allow you to submit code that does not compile. If your code doesn't compile, you will receive no credit for the assignment.