# Homework 2: Higher-Order Functions

In this assignment, you'll write several higher-order functions over lists.

**Restrictions**: You must not use Scala's builtin methods. In particular, you must not use any of the methods on Scala's lists. However, feel free to use any of the list-processing functions that we've written in class or the code you wrote for the previous assignment. You may not use imperative features, such as `while` and `var`. You should also write any helper functions you think are necessary. You may also use functions you write for one part of the assignment to solve another part of the assignment.

## Preliminaries

You should create a directory-tree that looks like this:

```
./hof
├─project
│ └─plugins.sbt
└─src
  ├─main
  │ └─scala. ........................................ Your solution goes here
  └─test
    └─scala.......................................... Yours tests go here
```

The `project/plugins.sbt` file must have exactly this line:

```
addSbtPlugin("edu.umass.cs" % "compsci220" % "1.0.0")
```

## Exercises

Within an object called `HOF`, implement the following functions. (You may use the template in fig. 6.1)

1. Write the `map2` function, which maps over two lists:

   ```
   def map2[A,B,C](f: (A, B) => C, alist1: List[A], alist2: List[B]): List[C]

   test("map2 with add") {
     def add(x: Int, y: Int): Int = x + y
     assert(map2(add, List(1, 2, 3), List(4, 5, 6)) == List(5, 7, 9))
   }
   ```

```
object HOF {

  def map2[A,B,C](f: (A, B) => C, lst1: List[A], lst2: List[B]): List[C] = ???
  def zip[A,B](lst1: List[A], lst2: List[B]): List[(A, B)] = ???
  def flatten[A](lst: List[List[A]]): List[A] = ???
  def flatten3[A](lst: List[List[List[A]]]): List[A] = ???
  def buildList[A](length: Int, f: Int => A): List[A] = ???
  def mapList[A, B](lst: List[A], f: A => List[B]): List[B] =
  def partition[A](f: A => Boolean, lst: List[A]): (List[A], List[A]) = ???
  def merge[A](lessThan: (A, A) => Boolean, alist1: List[A], alist2: List[A]): List[A] = ???
  def sort[A](lessThan: (A, A) => Boolean, alist: List[A]): List[A] = ???

}
```

Figure 6.1: Solution template.

You may assume that `alist1` and `alist2` have the same length (i.e., do whatever you think is reasonable).

2. Scala lets you write `(x, y)` to create *tuples*. For example:

   ```
   val nameSsn: (String, Int) = ("John Smith", 19970293)
   val emailAdmin: (String, Boolean) = ("carberry@cs.umass.edu", true)
   val xy: (Double, Double) = (3.0, 4.0)
   ```

   Tuples are a convenient way to package two related values together, without having to create a new type to hold them. You can use pattern matching to extract the components of tuples, or simply write `xy._1` and `xy._2` to access the first and second component respectively.

   Write the `zip` function, which tuples corresponding elements in a list:

   ```
   def zip[A,B](alist1: List[A], alist2: List[B]): List[(A, B)]

   test("zip test 1") {
     assert(zip(List(1, 2, 3), List(4, 5, 6)) == List((1,4), (2, 5), (3, 6)))
   }

   test("zip test 2") {
     assert(zip(List("George", "Teddy"), List("Washington", "Roosevelt")) ==
            List(("George", "Washington"), ("Teddy", "Roosevelt")))
   }
   ```

3. Write the function `flatten`, which flattens a nested list:

   ```
   def flatten[A](alist: List[List[A]]): List[A]

   test("flatten test") {
     assert(flatten(List(List(1, 2), List(3, 4))) == List(1, 2, 3, 4))
   }
   ```

4. Write the `flatten3` function, which flattens a triple-nested list:

   ```
   def flatten3[A](alist: List[List[List[A]]]): List[A]
   ```

5. Write the `buildList` function, which builds a list of the given length. Each element is determined by applying `f` to the index of the element:

   ```
   def buildList[A](length: Int, f: Int => A): List[A]

   test("buildList test") {
     def f(x: Int) = x
     assert(buildList(10, f) == List(0, 1, 2, 3, 4, 5, 6, 7, 8, 9))
   }
   ```

6. Write the `mapList` function, which maps each element to a list and returns the list of all results:

   ```
   def mapList[A, B](alist: List[A], f: A => List[B]): List[B]

   test("mapList test") {
     def f(n: Int): List[Int] = buildList(n, (_: Int) => n)
     assert(mapList(List(1, 2, 3), f) == List(1, 2, 2, 3, 3, 3))
   }
   ```

7. Write the `partition(f, alist)` function, which splits a list into two sub-lists, where the first list has all the elements of `alist` on which `f` produces `true` and the second list has all the elements of `alist` on which `f` produces `false`.

   The ordering of elements in the two sub-lists should be the same as in `alist`.

   ```
   def partition[A](f: A => Boolean, alist: List[A]): (List[A], List[A])

   def isEven(n: Int): Boolean = n % 2 == 0

   test("partition test 1") {
     assert(partition(isEven, List(1,2,3,4,5,6)) == (List(2,4,6), List(1,3,5)))
   }

   test("partition test 2") {
   ```

```
  assert(partition(isEven, List(2,4,6)) == (List(2,4,6), Nil))

test("partition test 3") {
  assert(partition(isEven, List(1,3,5)) == (Nil, List(1,3,5)))
}
```

8. Write the `merge(lessThan, alist1, alist2)` function, which merges the two elements of the two lists. You should assume that `alist1` and `alist2` are sorted by the ordering defined by `lessThan`. the produced list should also be sorted by the same ordering.

```
def merge[A](lessThan: (A, A) => Boolean, alist1: List[A], alist2: List[A]): List[A]

def lt(x: Int, y: Int): Boolean = x < y

test("merge test 1") {
  assert(merge(lt, List(5, 3, 1), List(10, 6, 0)) == List(10, 6, 5, 3, 1, 0))
}
```

9. Write the `sort(lessThan, alist)` function, which sorts `alist` by the ordering defined by `lessThan`.

```
def sort[A](lessThan: (A, A) => Boolean, alist: List[A]): List[A]


test("sort test 1") {
  assert(sort(lt, List(5,1,2,3,4,5)) == List(5,5,4,3,2,1))
}
```

## 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.