22c:022 Object-Oriented Software Development
Fall 2013

Homework 1

Due: Thursday, Sep 12 by 7pm

This is a programming assignment in Scala to be done *individually*. Type your solutions in a *plain text file*[1] called `hw1.scala` and submit it through the Dropbox section of ICON. You can use the template `hw1.scala` file on ICON. Make sure you *write your name in that file*.

Each of your answers *must be free of static errors*. You may receive no credit for problems whose code contains syntax or type errors.

Pay close attention to the specification of each problem and the restrictions imposed on its solution. Solutions ignoring the restrictions may receive no credit.

1 Scala Programming Imperative Style

In the following problems *do not* define methods recursively. Instead, use mutable variables and while or for loops as needed in method definitions. You may use auxiliary methods if needed.

1. Write a Scala method `mul: (m: Int, n: Int) Int` which, given two *non-negative* integers m and n,[2] returns the result of the integer multiplication of m by n. To implement `mul` do not use any predefined arithmetic operators in Scala other than `>`, `+` and `-`.

2. Write a Scala method `concatAll: (l: List[String]) String` which, given any list l of strings, returns the concatenation of all the strings in l. For example, `concatAll(List("un", "be", "liev", "a", "ble"))` is `"unbelievable"`.

   Note that in Scala `""` is the empty string and `++` is the infix concatenation operator for strings.

3. Write a Scala method `isIn: (x: Int, l: List[Int]) Boolean` which, given an integer value x and an integer list l, returns `true` if and only if x is an element of l. *Do not* use the `contains` method or similar methods of the `List` class or any other class.

4. Write a Scala method `squareAll: (l: List[Int]) List[Int]` which, given an integer list l, "squares" every element in it, that is, returns a new list containing (in the same order) the square of each element of l. For instance, `squareAll(List(1,2,-3,0))` is `List(1,4,9,0)`.

[1] MS Word files are *not* plain text files.

[2] Here or later, a specification such as "given a non-negative integer" does not mean that you must check that the input is non-negative. To the contrary, it means that you *do not* need to check it, you can just assume it. What happens when the input is negative is undefined.

2 Scala Programming Functional Style

In the following problems *do not* use any mutable variables (i.e., variables declared with `var`) or any mutable predefined types such as arrays, mutable sets, and so on. You may define methods recursively this time. *Do not* use the `if` operator in Problems 1-4 below.

1. Redo Problem 1 (on `mul`) in Part 1, but now under the restrictions above.
2. Redo Problem 2 (`concatAll`) under the restrictions above.
3. Redo Problem 3 (`isIn`) under the restrictions above.
4. Redo Problem 4 (`squareAll`) under the restrictions above.
5. Write a Scala method `remove: (x: Int, l: List[Int]) List[Int]` which, given an integer value x and an integer list l, "removes" all the occurrences of x from l, that is, returns a list containing (in the same order) all and only the elements of l that differ from x.

   For example, `remove(2, List(2,1,3,3,2,1))` is `List(1,3,3,1)`,
   `remove(5, List(2,1,3,3,2,1))` is `List(2,1,3,3,2,1)`.
6. Consider the following Scala methods. Analyze their code and their behavior on a few input examples to figure out what they do. For `m` assume that both input lists are ordered non-decreasingly.

```
def m(l1:List[Int], l2:List[Int]):List[Int] =
 (l1, l2) match {
  case (Nil, _) => l2
  case (_, Nil) => l1
  case (h1 :: t1, h2 :: t2) => {
   if (h1 < h2)
    h1 :: m(t1, l2)
   else
    h2 :: m(l1, t2)
  }
 }

def s(l:List[Int]) = {
 def s_aux(l0:List[Int], l1:List[Int], l2:List[Int]):(List[Int], List[Int]) =
  l0 match {
   case Nil => (l1, l2)
   case x :: Nil => (x :: l1, l2)
   case x1 :: x2 :: t => s_aux(t, x1 :: l1, x2 :: l2)
  }

 s_aux(l, Nil, Nil)
}
```

   For each method write in a Scala comment a concise description in English of its functionality, along the lines of the descriptions provided for the methods in the previous problems (given this and that, it returns so and so). *Just describe the input-output behavior of the method, not the internal algorithm.*

7. Write a Scala method `pair: (list1: List[Int], list2: List[Int]) List[(Int,Int)]` which "pairs" two lists of integers of the same length; that is, produces a list of pairs of corresponding elements in the input lists. For example, `pair(List(4,2,5,9),List(3,5,6,11))` is `List((4,3),(2,5),(5,6),(9,11))`.[3]

8. Recall the basic recursive algorithm for merge sort, as applied to lists of integers.

   - Partition somehow the elements of the current list into two sublists of similar size,
   - sort the two sublists separately,
   - merge the two sorted sublists into a single sorted one.

   Write a recursive Scala method `mergeSort: (l: List[Int]) List[Int]` implementing the algorithm above to sort its input list non-decreasingly. For example, `mergeSort(List(1,3,7,1,4,5))` should evaluate to `List(1,1,3,4,5,7)`. Note that the empty list and all singleton lists are always sorted.

   You can use as needed any methods from previous problems in this assignment as helper methods for `mergeSort`.

[3] Note again that, according to the specification, you need not worry about what happens when the input lists have different lengths.