Project 2 (10 points)

Getting started with Scala

Goals

In this course, we will explore using Scala and an actors model to program concurrent computations. However, the Scala programming language is not a major teaching topic in the course schedule. Instead, you are expected to begin to learn Scala by going through the tutorials, discussing with your classmates and finishing this lab. After completing this lab, you should gain knowledge of Scala programming, its syntax and basic programming conventions as preparation for using an actors model and its corresponding language features to program with concurrency.

Due Date

March 7th, 2019 (Thursday) 2:00 pm.

Late Penalty

As described in the course policies.

Collaboration Policy

This exercise may be performed individually or in pairs. You may also discuss general strategies with your classmates. However, you must complete coding on your own or within your pair. **No copying of or detailed line by line assistance on code is allowed.**

You are encouraged to look at the Scala API documentation while solving this exercise, which can be found here: http://www.scala-lang.org/api/current/index.html.

Note that Scala uses the String from Java, therefore the documentation for strings is found in the Javadoc API: http://docs.oracle.com/javase/6/docs/api/java/lang/String.html

Lab Description

Step 1: Project Set Up

Please create a project (a directory) named "Project_Two". Create an object "Main" in the default package (a file in the directory, in this context), then copy and paste the codes given in **`Main.scala`**, which sets up the input and output for functions and classes defined in step 2 to step 4.

The main function defined in **`Main.scala`** takes a series of parameters separated by white space:

`args(0)` – the number of lines of the Pascal triangle to print
`args(1)` – the amount in cents for which to calculate the number of combinations of change

`args(2), args(3)` – parenthesized expressions to be validated.

sample series of parameters is:
`10 50 (x+(y+4)) :-)(`

which will make the program print 10 lines of Pascal’s triangle, count the number of ways to make change of 50¢, verify whether parentheses are balanced in “(x+(y+4))” and “:-)(” expressions.

You may work at the command line using an editor such as vi , scalac to compile and scala to run. For example to execute using the above command line parameters you would type:

```
scala Main 10 50 “(x+(y+4)” “:-)(“
```

Alternatively, you may use the IDE of your choice. **However, the code that you submit for grading should compile & run through command line interactions and should not refer to any packages not included in the submission.**

Expected output:

```
step 2: print function
      *
     ***
    *****
   *******
  *********

  step 3: recursive algorithm
  Pascal's Triangle
  1
  1 1
  1 2 1
  1 3 3 1
  1 4 6 4 1
  1 5 10 10 5 1
  1 6 15 20 15 6 1
  1 7 21 35 35 21 7 1
  1 8 28 56 70 56 28 8 1
  1 9 36 84 126 126 84 36 9 1
  1 10 45 120 210 252 210 120 45 10 1

  Number of ways to make change of 50 cents is: 146

  expression: (x-(y+4)) is balanced
  expression: :-)( is not balanced
```

Step 2: A Simple Print Function

In the Main object defined in step 1, redefine the method `myPrint()` to print out the following output:

```
    *
   ***
  *****
 *******
*********
```

Step 3: Recursive Algorithms

Create a new object Recursion (file) in the same default package (directory) as Main. Copy the codes in **Recursion.scala**.

3.1. Pascal's Triangle

The following pattern of numbers is called *Pascal's* Triangle. The numbers at the edge of the triangle are all 1 and each number inside the triangle is the sum of the two numbers above it. Write a function that computes elements of Pascal's Triangle by means of a recursive process.

```
                1
               1 1
              1 2 1
             1 3 3 1
            1 4 6 4 1
          1 5 10 10 5 1
        1 6 15 20 15 6 1
      1 7 21 35 35 21 7 1
    1 8 28 56 70 56 28 8 1
```

Finish this exercise by implementing the `pascal(c: Int, r:Int): Int` function, which takes a column c and a row r, counting from 0, and returns the number at that spot in the triangle. For example, `pascal(0,2)=1`, `pascal(1,2)=2`, `pascal(1,3)=3`.

NOTE: the actual display of a Pascal Triangle won't be nicely centered. Rather, it will be left justified:

```
1
1 1
1 2 1
 … and so on.
```

3.2. Counting Change

Write a recursive function that counts how many different ways you can make change for an amount, given a list of coin denominations. For example, there are 3 ways to give change for 10¢ if you have coins with denomination 1¢, 5¢, and 10¢:

- 1 of 10¢
- 10 of 1¢
- 1 of 5¢ and 5 of 1¢
- 2 of 5¢

Finish this exercise by implementing the `countChange(money: Int, coins: List[Int]): Int` function. This function takes an amount to change, and a list of denominations for the coins.

Hint: Three functions of List type in Scala might be helpful:

`coins.isEmpty(): Boolean` – returns whether the list is empty
`coins.head(): Int` – returns the first element of the list
`coins.tail(): List[Int]` – returns the list without the first element

3.3. Parenthesis Balancing

Design a recursive algorithm to verify the balancing of parenthesis in a statement. For example, the following two statements have balanced parenthesis:

- (if(zero?x)max(/1x))
- ((x+3)/(y-4)+33)%2

The following statements do **NOT** have balanced parenthesis:

- :-)
- ())(

Notice that the last example above showed that only counting the number of left and right parenthesis is not enough for verification.

Finish this exercise by completing the implementation of the
`balance(chars:List[Char]): Boolean`
function.

Hint: The `isEmpty`, `head`, `tail` functions may still be useful. You could define an inner function if you want to pass extra parameters to the `balance` function

Step 4:

Create a readme file which includes your name(s) and the course number (CPSC 4820 or 6820) you enrolled in. You should also put any necessary information for grading in this file too. Put **Readme**, **Main.scala**, and **Recursion.scala** into one folder, zip it up and submit the zip file via Canvas.

Step 5:

Complete the associated questions/survey on Canvas.