Assignment AEC1 - Recursion

CSCI 242 - Computer Science II :: Version 1.0.0

Due date is posted in the LMS.

This is an extra credit assignment and will be worth 25% of a standard assignment. However, the "Specification" section of the programming rubric will be increased from 30 points to 40 points. To calculate your grade, your instructor will grade as usual (with the added 10 points), then multiply the result by .25.

Please read the entire assignment description before beginning the project.

Objective

The objective of this assignment is to reinforce the concept of recursion in Java.

To accomplish this, you will implement a number of recursive methods that are independent of each other but each reinforces the concepts of recursion.

Introduction

"This is like déjà vu all over again."
--Yogi Berra

The Assignment

This assignment will be a problem set consisting of several small problems to solve in isolation. For each problem, you are to write a short recursive method (or methods) to solve it. By short, we mean that none of them will require more than 20 lines to complete. That does not mean you should put this assignment off until the last minute though — recursive solutions can often be formulated in just a few concise, elegant lines but if you do not yet comprehend recursion, they can be extremely dense and complex.

A few things to take note of:

1. Since this is an extra credit assignment, the help from your instructor will be limited… Obviously, your instructor will provide clarification if needed but help with the solution will be limited.
2. You do not need to complete all of the small, isolated problems to get some points. Note that each problem has points associated with it, so do the problems you can!
3. Test, test, test. It is okay to leave your testing code in the assignment – in fact, *we want to see your testing code*!
4. Do NOT use instance or static variables. Pass all information using parameters.
5. Follow directions carefully.
   a. For each exercise, we specify the method signature. Your method must exactly match that signature (same name, same arguments, and same return type). Your instructor will be providing their own test `main`, so failing to use the proper signature will result in the inability to test your code and a 0 on that problem.
   b. The method signatures described above and shown below are all `static`. This is because you should have one "main driver" with all of the recursive methods in the "main driver". In other words, put the `main()` and the recursive methods in one file.
   c. Your methods must use recursion; even if you can come up with an iterative alternative, we insist on a recursive formulation!

The Project

You are required to use IntelliJ IDEA for your assignment. The following is your project information:

Package:	`edu.uwp.cs.csci242.assignments.ec1.recursion`
IntelliJ Project Name:	`AEC1_Recursion`
Zip filename:	`AEC1.zip`

Method 1 – String Cleaning

This method counts for 5 points of the 40 points in the "Specification" category of the Programming Rubric. The following is the signature of the recursive method:

```
public static String stringClean ( String str )
```

Given a string, return, recursively, a "cleaned" string where adjacent characters that are the same have been reduced to a single character. So, for example, the string `yyzzza` yields `yza`.

```
stringClean("yyzzza") → "yza"

stringClean("abbbcdd") → "abcd"

stringClean("Hello") → "Helo"
```

Method 2 – Count Digit

This method counts for 5 points of the 40 points in the "Specification" category of the Programming Rubric. The following is the signature of the recursive method:

```
public static int countDigit ( int num, int digit )
```

Write a recursive function `countDigit()` to count the number of time a particular digit appears in a number *n*, where *n* > 0. Hint: process the digits right to left. It is easy to see the least significant digit using the mod (%) operator.

```
countDigit ( 222, 2 )      → 3

countDigit ( 123414, 1 )   → 2

countDigit ( 123414, 5 )   → 0
```

Method 3 – Balancing Parentheses

This method counts for 10 points of the 40 points in the "Specification" category of the Programming Rubric. The following is the signature of the recursive method:

```
public static boolean isBalanced ( String str )
```

In the syntax of most programming languages, there are characters that occur only in nested pairs, which are called bracketing operators. Java, for example, has these bracketing operators:

```
( . . . )

[ . . . ]

{ . . . }
```

In a properly formed program, these characters will be properly nested and matched. To determine whether this condition holds for a particular program, you can ignore all the other characters and look simply at the pattern formed by the parentheses, brackets, and braces. In a legal configuration, all the operators match up correctly, as shown in the following example:

```
{ ( [ ] ) ( [ ( ) ] ) }
```

The following configurations, however, are illegal for the reasons stated:

`( ( [ ] )` → The line is missing a close parenthesis.

`) (` → The close parenthesis comes before the open parenthesis.

`{ ( } )` → The parentheses and braces are improperly nested.

For this problem, your task is to write a recursive function that takes a string from which all characters except the bracketing operators have been removed. The function should return true if the bracketing operators in the string are balanced, which means that they are correctly nested and aligned. If the string is not balanced, the function returns false.

Although there are many other ways to implement this operation, you should code your solution so that it embodies the recursive insight that a string consisting only of bracketing characters is balanced if and only if one of the following conditions holds:

- The string is empty.
- The string contains "()", "[]", or "{}" as a substring and is balanced if you remove that substring.

For example, the string "[(){}]" is shown to be balanced by the following chain of calls:

```
isBalanced("[(){}]") →
   isBalanced("[{}]") →
      isBalanced("[]") →
         isBalanced("") → true
```

Method 4 – Split Array

This method counts for 10 points of the 40 points in the "Specification" category of the Programming Rubric. The following is the signature of the recursive method:

```
public static boolean splitArray(int [] array)
```

Given an array of `int`s, is it possible to divide the `int`s into two groups, so that the sums of the two groups are the same. Every `int` must be in one group or the other. Write a recursive helper method that takes four parameters, the array, an index into the array, and two partial sums. Make the initial call to your recursive helper from `splitArray()`. Your helper will make two recursive calls, one where the current element is added onto the first sum, and one where it is added onto the second sum. If either returns true, the method returns true.

```
splitArray([2, 2]) → true
splitArray([2, 3]) → false
splitArray([5, 2, 3]) → true
splitArray([2, 5, 3]) → true
```

Method 5 – Tricky Towers of Hanoi

This method counts for 10 points of the 40 points in the "Specification" category of the Programming Rubric. The following is the signature of the recursive method:

```
public static void trickyHanoi ( int disks )
```

The book discusses the Towers of Hanoi problem, where the goal of the "game" was to transfer all the disks from peg A to peg C. This problem is a variation on that puzzle. We are to transfer all the disks from peg A to

peg C, but every move must involve peg B (either as the from peg or the to peg). We still maintain the restriction that only smaller disks can go on top of larger disks.

Examples:

```
trickyHanoi ( 1 )
Move disk from peg A to peg B.
Move disk from peg B to peg C.

trickyHanoi ( 2 )
Move disk from peg A to peg B.
Move disk from peg B to peg C.
Move disk from peg A to peg B.
Move disk from peg C to peg B.
Move disk from peg B to peg A.
Move disk from peg B to peg C.
Move disk from peg A to peg B.
Move disk from peg B to peg C.
```

Submission

Zip up your entire project folder into a single zip file named `AEC1.zip` and submit that one file. Submit the Zip file to the LMS "dropbox" labeled `Assignment EC1 – Recursion >>> DROPBOX`. Your instructor should be able to unzip the entire project and load the entire project their IDE without incident.

Grading

The LMS has a rubric attached to the assignment. This rubric is similar to the one shown in the document titled, "Programming Requirements and Rubric" found in the LMS.

Good software engineering is expected. Remember to follow the Coding Guidelines 1.0.0 found in the LMS. Use lots of comments throughout the code, lots of methods, and appropriate variable names. Make sure you validate your input.

Ω