Lab 7: Recursion [noun]: a procedure that uses recursion.
CSCI 2101 – Fall 2017

Due: Thursday, November 9, 11:59 pm
Collaboration Policy: Level 1
Group Policy: Individual

This week's lab will give you a refresher on writing recursive algorithms. Recursion can be difficult to wrap your head around at first, so we will focus so some smaller recursive problems first. Next week, you will move onto a larger lab that also uses recursion.

Owing to next week's exam, this lab is somewhat shorter than most past labs. Your task is to complete a set of recursive methods. As you work through these problems, remember the essential rules of all recursive algorithms:

- You must have (at least one) base case, which does *not* use recursion.
- You must have (at least one) recursive case, which *does* use recursion.
- Each recursive case must move you *towards* a base case.
- You should not write any loops (if you are, you're probably writing an iterative solution rather than a recursive solution).

Good recursive solutions tend to be short, compact, and elegant – but they may be tricky to come up with!

1 Recursive Methods

Each of the four methods below should be written in a single Java file named `Recursion.java`. Since no regular classes will be defined here, each of the methods will be a `static` method that is called directly from your `main` method. You must declare your methods exactly as specified for each method below. Additionally, you should not need to declare any instance or static variables.

Write test code in a `main` method to verify that your recursive methods are working (and be sure to try more than one possible input).

1.1 Cannonballs

Imagine stacking cannonballs in a pyramid – e.g., there's one cannonball at the top, sitting on top of a level composed of four cannonballs, sitting on top of a level composed of nine cannonballs, and so forth. Write a recursive method `countCannonballs` that takes a non-negative integer representing the height of the pyramid (i.e., number of levels) and returns the number of cannonballs in the pyramid. Here is the required method definition:

```
public static int countCannonballs(int height)
```

You may find it useful to note that each level of the pyramid consists of a square of cannonballs. You should not need to call any methods except for your recursive calls.

1.2 Digit Sum

Write a recursive method `digitSum` that takes a non-negative integer and returns the sum of its digits. For example, `digitSum(1234)` returns $1+2+3+4 = 10$. You are *not* allowed to convert the number to a `String`, and instead must only work with the value numerically. Here is the required method definition:

```
public static int digitSum(int n)
```

You may find the mod (%) and division operators useful here. You should not need to call any methods except for your recursive calls.

1.3 Binary Numbers (redux)

Recall from the Two Towers lab that a binary number (base-2) is a number made up of bits, each of which is 0 or 1 only, in which each bit represents a power of 2. A binary number like 10011 is converted to a decimal number (base-10) by multiplying the bits by the powers of 2 and adding them. In the case of 10011, reading from left to right, $1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 19$.

Suppose we want to write an algorithm to convert a series of bits to its decimal value. You could easily write an iterative algorithm that does this by looping over the bit values and powers of 2 and adding them together.

You can also approach this problem recursively, however. Consider the effect of adding an extra bit to the right side of an existing binary number. For example, suppose you have the binary number 11 (decimal value 3). Adding an extra zero bit simply doubles the value (binary 110, decimal 6). Adding an extra one bit also doubles the value, but then adds one to the end result (binary 111, decimal 7). Try adding a bit to another binary number or two to convince yourself that this property always holds.

Using this property, write a recursive method `bitsToDecimal` that is given a series of bits as a `String` and returns the decimal value of the binary number specified. For example, calling `bitsToDecimal("101")` should return 5. Here is the required method definition:

```
public static int bitsToDecimal(String bits)
```

You may not call any `String` methods except `equals`, `length`, `substring`, and `charAt`. These restrictions are to ensure that you write a truly recursive solution and do not rely on iterative `String` methods.

One gotcha to watch out for is how you check for whether a specific `char` is a 1 bit or a 0 bit. In particular, Java will let you compare a `char` with an `int`, but the `char` of a particular number is *not* equal to its integer value. For example:

```
char bit = '1'; // this is a char: it has a numeric value, but its value is not 1
boolean test1 = (bit == '1'); // true, char comparison
boolean test2 = (bit == 1); // FALSE, the int 1 is not equal to the char '1'
```

It's easy to write the latter when you really want the former (since the compiler won't complain even though you're comparing different types).

1.4 Palindromes

A *palindrome* is a word that is the same spelled forwards or backwards. For example, the word "racecar" is a palindrome, but the word "car" is not. Palindromes can also be multiple words, e.g., "rise to vote sir" (note that in multi-word palindromes, we normally ignore the spaces).

Write a recursive method `isPalindrome` that accepts a `String` and returns whether that string represents a palindrome. You may assume that the string consists only of letters and spaces. Your method may be case sensitive (so "racecar" would be considered a palindrome, but "Racecar" would not). However, you should *ignore* spaces; so for example, the string "rise to vote sir" would be considered a palindrome. Here is the required method definition:

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

As in the previous method, you may not call any `String` methods except `equals`, `length`, `substring`, and `charAt`. In particular, this means that you cannot use `replace` to remove spaces in the string.

2 Evaluation

As usual, your completed program will be graded on correctness, design, and style. Make sure that your program is documented appropriately and is in compliance with the style guide. Lastly, make sure that your name is included in your Java file.

3 Submitting Your Program

Submit your program on Blackboard in the usual way. Remember to create a zip file named with your username(s) and lab number, e.g., `sbowdoin-lab7.zip`, and upload that file.