

CS 272 Homework 4 - Recursion

In Google Drive, go to **File->Make a Copy** and save this as **h03_YourLastName_YourFirstName**. You will not be able to edit the shared master copy. Complete each of the homework exercises below using Eclipse.

There are two kinds of deliverables:

1. When you are finished, make a **ZIP** file out of the entire eclipse project folder and upload to Canvas.
2. Save this document as a **PDF** and submit the PDF to Canvas.

For **Part A** and **Part B**, you will download the [starter code](#) (containing an Eclipse project) and complete it. The tests are already there.

H04-A—String Finder

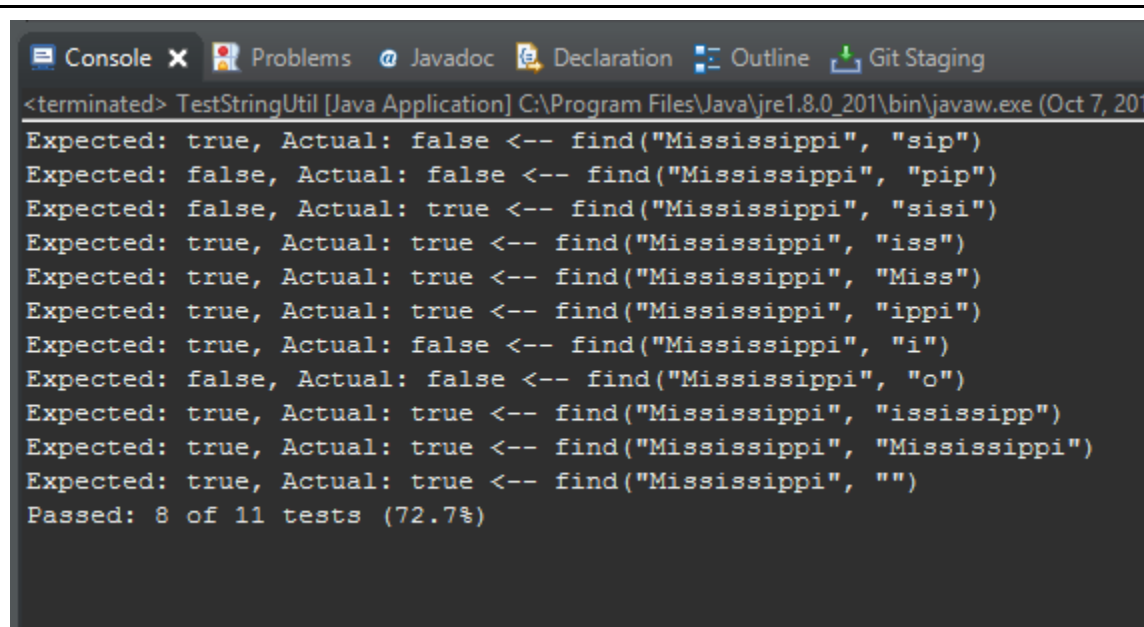
Use recursion to implement a method

```
public static boolean find(String text, String str)
```

that tests whether a given text contains a string. For example, `find("Mississippi", "sip")` returns true.

Hint: If the text starts with the string you want to match, then you are done. If not, consider the text that you obtain by removing the first character.

Test your method by running **TestStringUtil**.



```
<terminated> TestStringUtil [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (Oct 7, 2017)
Expected: true, Actual: false <-- find("Mississippi", "sip")
Expected: false, Actual: false <-- find("Mississippi", "pip")
Expected: false, Actual: true <-- find("Mississippi", "sis")
Expected: true, Actual: true <-- find("Mississippi", "iss")
Expected: true, Actual: true <-- find("Mississippi", "Miss")
Expected: true, Actual: true <-- find("Mississippi", "ippi")
Expected: true, Actual: false <-- find("Mississippi", "i")
Expected: false, Actual: false <-- find("Mississippi", "o")
Expected: true, Actual: true <-- find("Mississippi", "ississipp")
Expected: true, Actual: true <-- find("Mississippi", "Mississippi")
Expected: true, Actual: true <-- find("Mississippi", "")
Passed: 8 of 11 tests (72.7%)
```

H04-B—Square Root Computer

The following method was known to the ancient Greeks for computing square roots. Given a value $x > 0$ and a guess g for the square root, a better guess is $(g + x/g) / 2$. Write a recursive helper method `public static squareRootGuess(double x, double g)`. If g^2 is approximately equal to x , return g , otherwise, return `squareRootGuess` with the better guess. Then write a method `public static squareRoot(double x)` that uses the helper method.

Test your method by running **TestMyMath**.

```
<terminated> TestMyMath [Java Application] C:\Program Files\Java\jre1.8.0_20
MyMath.sqrt(25.0) = 5.000000000052309 PASS
  Math.sqrt(20.0) = 4.47213595499958
MyMath.sqrt(20.0) = 4.47213595500155 PASS
  Math.sqrt(10.0) = 3.1622776601683795
MyMath.sqrt(10.0) = 3.1622776650643187 PASS
  Math.sqrt(5.0) = 2.23606797749979
MyMath.sqrt(5.0) = 2.236068879361104 PASS
  Math.sqrt(2.0) = 1.4142135623730951
MyMath.sqrt(2.0) = 1.4142156387086517 PASS
  Math.sqrt(1.0) = 1.0
MyMath.sqrt(1.0) = 1.0000004995004996 PASS
  Math.sqrt(0.5) = 0.7071067811865476
MyMath.sqrt(0.5) = 0.7071078673913538 PASS
Passed: 7 of 7 tests (100.0%)
```

H04-C—Your Choice

For Part C, you may work with a partner. Write both you and your partner's names in the source files.

Choose one of the following problems, implement it and test it. Similar to Parts A & B, you should put your class files in the `occ.cs272.h04` package and your tester program in the default package.

For the problem you plan to solve:

1. What type of recursion do you think is needed?
2. Describe the general strategy and the terminating cases. Explain the steps you plan to use.

StairCase

Suppose you want to climb a staircase with n steps and you can take either one or two steps at a time. Recursively enumerate all paths. For example, if n is 5, the possible paths are:

[1, 2, 3, 4, 5], [1, 3, 4, 5], [1, 2, 4, 5], [1, 2, 3, 5], [1, 3, 5]

BillsToPay

Given an integer price, list all possible ways of paying for it with \$100, \$20, \$5, and \$1 bills, using recursion. Don't list duplicates.

PhoneWords

Phone numbers and PIN codes can be easier to remember when you find words that spell out the number on a standard phone pad. For example, instead of remembering the combination 5282, you can just think of JAVA.

Write a recursive method that, given a number, yields all possible spellings (which may or may not be real words).

With the help of your method, try to find a good way to remember the number 587846.

For PhoneWords, I would use a permutation where the simplest input would be an empty string. I would then create a switch statement where each number has the associated letters attached to it in an array. As it loops down to the smallest/empty string, as it goes back up, it will go through each case and pull out a letter from the array of letters. I would need more time to find out a way to make sure duplicates were not utilized in this case. Maybe an index of some sort to state where I am in the case?

Don't forget to upload the zip file of your project along with the pdf of this document to Canvas.