# Lab 3

Due: 11:00pm on October 5, 2014

40 points

We will step away from your résumé for this lab and focus on creating PHP code. Be sure to follow the directions below. In order to find your file on the server, make sure you specify the URL to access your file in your d2l submission as a comment. It should be stored at http://webdev.cs.uwosh.edu/userID/Labs/lab3.php. A link from the comments in your d2l submission makes your page much easier to find. Because this lab is considerably more difficult than previous labs, it will be worth twice as much and you will be given 2 lab sessions to complete it.

1. Create a file called *lab3.php* and create all of your functions in this file. When finished, upload this file to your Labs folder on the webdev server. You must also submit your lab3.php file to the lab 3 dropbox on d2l for testing.

2. The output of your page should be a valid HTML document. Be sure to add the HTML validator code at the bottom of your page.

3. Create a *main* function that you will use to test all of your functions. Make 1 call to the main function in your PHP file. We will be using our own main function to test your code so make sure your functions create self-containing HTML code that verifies. Don't rely on your main method to create any HTML code as it will be deleted. Your main function should have no arguments. Make sure you test each of your functions thoroughly in your main function.

4. Be sure your functions are capitalized and named exactly as they are stated.

5. For all of your functions, if you are producing any output, you must not using any PHP print or echo calls. Use embedded PHP expressions to achieve your output.

6. Create a function called **fibonacci** that accepts a number **n** as an argument and returns the $n^{th}$ Fibonacci number. For this problem, fibonacci(0) = 0, fibonacci(1) = 1 and fibonacci(n) = fibonacci(n-1) + fibonacci(n-2). If you hate recursion, you are welcome to use the closed form to get your answer. But if you hate recursion, you are not going to like the last 2 functions. This function need not print anything out. You only need to return a value.

7. Create a function called **isoceles** that reads in from a file called *triangle.txt* and creates an array of IsocelesTriangles. You must create IsocelesTriangle objects here and therefore must create an IsocelesTriangle class. The file will be formatted such that each line contains 1 triangle. The first element of the line will be the base, the second element will be the height. You can assume the input file is properly formatted but cannot assume a minimum or maximum number of triangles. Once you have read in all triangles, print out the one that has the largest area. Print out the one that has the largest perimeter (these are isoceles triangles, you can use geometry to figure out the lengths of the other 2 sides). Print out the triangle with the smallest area. Print out the triangle with the smallest perimeter. You may want to create auxilary functions to accomplish these tasks. This function doesn't return anything.

8. Create a function called **stringTable** that accepts an array of strings as the argument. This function will print out a table consisting of the number of occurrences of each letter in each word. Only the letters that occur in one of the words should show up in the output. For this problem, case doesn't matter (h is the same as H). You can assume strings will only be letters. For instance, if the following array were passed in: {"hello", "How", "are", "you" }, the following table would be output:

| | hello | how | are | you |
|---|---|---|---|---|
| a | 0 | 0 | 1 | 0 |
| e | 1 | 0 | 1 | 0 |
| h | 1 | 1 | 0 | 0 |
| l | 2 | 0 | 0 | 0 |
| o | 1 | 1 | 0 | 1 |
| r | 0 | 0 | 1 | 0 |
| u | 0 | 0 | 0 | 1 |
| y | 0 | 0 | 0 | 1 |

   The formatting of the table needs to be exactly as it is above. The order the strings appear in the array is the same order they appear in the first row. The letters in the first column need to be in alphabetical order. This function should not return anything.

9. Create a function called **subsetMultiples** that accepts an array of integers and a single integer we'll call target. The function returns true if there exists a subset of the array whose product is equal to the target, false otherwise. In other words, consider the following array called myArr: {2, 3, 5, 6}. The following function call would return true: subsetMultiples(myArr, 15). This is because the subset {3, 5} is 3 * 5 = 15. The following function call would return false: subsetMultiples(myArr, 9) because no subset of {2, 3, 5, 6} has a product that equals 9. Let one of us know right away if you do not know what a subset is or do not understand this problem. This function need not print anything out. You only need to return true or false.

10. Create a function called **censorWord** that has 3 arguments: a word, a string that

is to be censored and a string that will replace all occurrences of the censored string. This function works by taking all of the instances of the censored string and replacing them with the replacement string. For example, if the function is called like this:
censorWord("chicago", "c", "r"))
the function should return be "rhirago". In other words, all of the c's were replaced with r's. If the function call were this:
censorWord("myfavoritestring", "i", "a")
the function would return "myfavoratestrang". In other words, all instances of "i" were replaced with "a". Lastly, if the function were called like this:
censorWord("cats dogs cats", "cats", "fish")
the function would return "fish dogs fish". Because this function would be trivial if you used the built-in replace functions, all built-in php replacement functions are not allowed. You may use other built-in functions like strlen and strcmp, but replacement functions are not allowed. Ask one of us if you are unsure if a function is allowed or not. This function need not print anything out. You only need to return the censored string.