Assignment #3
Decomposition/Functions
Due: Wednesday, 11/05/14, 11:59pm

Grading: For each programming assignment, you are graded by explaining and demoing your code to a TA. Your job is to convince the TA that your program works correctly, i.e. show your TA how to use/break your program. Your grade is based on Polya’s 4 steps for solving problems:

- Understanding the problem. (Recognizing what is asked.) (2.5 pts)
- Devising a plan. (Responding to what is asked.) (2.5 pts)
- Carrying out the plan. (Developing the result of the response.) (65 pts)
- Looking back. (Checking/Testing. What does the result tell me?) (10 pts)

Problem Statement: Write a C++ program from your design in Assignment #2. Expand the problem from Assignment #2 to contain functions for error checking, calculating the average, and prompting and getting input from the user. In this assignment, we are going to make a better grade calculator!!! You will ask the user how many tests, assignments, quizzes, labs, and if there is a final for the class. Based on these numbers, you will prompt the user for the scores for each of the categories to calculate the average for a specific class. Since we have categories, we need to know how much each category weighs in the percentage of the grade. For example, our class has the following percentages: labs – 10%, quizzes – 10%, assignments – 40%, tests – 25%, final – 15%.

Example Run:

    How many tests? 1
    How many quizzes? 2
    How many assignments? 2
    How many labs? 4
    Is there a final? 0

    Test 1 score: 100
    Quiz 1 score: 100
    Quiz 2 score: 100
    Assignment 1 score: 100
    Assignment 2 score: 100
    Lab 1 score: 100
    Lab 2 score: 100
    Lab 3 score: 100
    Lab 4 score: 100
Test weight: 25  
Quiz weight: 10  
Lab weight: 10  
Assignment weight: 40

Your percentage in this class is: 100

It is really important to re-design at this point, especially when writing functions. You need to modify your design from Assignment #2 to match these new requirements for the program. While you are redesigning, think about modularization of your code. The idea is to keep each function, including main, to a maximum of 10 lines, and this does not include blank lines. For example, here are some functions you might want to have:

- start_calculator();
- is_number();
- is_in_range();
- calculate_average();
- get_num_tests();

- In addition, create any other functions you might need to properly modularize your code to 10 lines in each function.

(10 pts) **Program Style/Comments**

In your implementation, make sure that you include a program header in your program, in addition to proper indentation/spacing and other comments! Below is an example header to include. Make sure you review the style guidelines for this class, and begin trying to follow them, i.e. don’t align everything on the left or put everything on one line! 

[http://classes.engr.oregonstate.edu/eecs/fall2014/cs161-001/161_style_guideline.pdf](http://classes.engr.oregonstate.edu/eecs/fall2014/cs161-001/161_style_guideline.pdf)

You are graded on having a **header, function headers with pre/post conditions, proper comments, and readable code with indentation** and vertical spacing that is CONSISTENT throughout your program. DO NOT align your entire program on the left side. This will cause you to automatically lose the full 10 points. In addition, do not forget your program header!!!

(10 pts) **Program Design for Assignment #4**

What if the user entered the input at the time when the program is executed without prompts? For example, you run your program like this:

```
./calc_grade -t 1 -q 2 -a 2 -l 4 -f 0
```

This runs your program with the example above 😊. Now, you don’t need to prompt, but you do need to process these options and values in any order, along with error handling.
We can get this input by defining parameters in the main function, i.e. `int main(int argc, char *argv[])`. The `argc` is the number of arguments supplied to the program, including its own name. In our example, this would be 11 arguments, and then the actual arguments are located to memory pointed to by `argv[]`. These values are C-style strings, and they can be accessed using `argv[0] – argv[10]`. How would this change your program, now that all the input is coming in as a C-style string? Look into the string.h library, or the cstring library. This library works with C-style strings, instead of C++ string objects.

Electronically submit your C++ program (.cpp file, not your executable!!!) and design document for Assignment #4, as a pdf, by the assignment due date, using TEACH.