Lab 3: Simple C Functions/Using Testing Framework

Due date: Friday, January 13, during the lab period.

(Note: you will get another assignment on Friday, January 13.)

Lab Assignment

Assignment Preparation

Lab type. This is an **individual lab**. Each student will submit his/her set of deliverables.

Collaboration. Students are allowed to consult their peers[1] in completing the lab. Any other collaboration activities will violate the *non-collaboration* agreement. No direct sharing of code is allowed.

Purpose. We are starting to work with the main concepts of C. This lab will let you continue developing functions and continue to master the craft of testing your code.

Programming Style. All submitted C programs must adhere to the programming style described in detail at

http://users.csc.calpoly.edu/∼cstaley/General/CStyle.htm

When graded, the programs will be checked for style. Any stylistic violations are subject to a 10% penalty. Significant stylistic violations, epsecially those that make grading harder, may yield stricter penalties.

Please note, that not all instructions from the link above are applicable to your Lab 2 assignments. If you do not understand the meaning of an instruction, please consult your instructor.

The following instructions are **in addition** to the rules outlined at the link above.

[1] A peer for the purpose of CPE 101 is defined as *"student taking the same section of CPE 101"*.

- **Short lines.** No line in your program shall be longer than 80 characters.
- **Header Comments.** The header comment **must be supplied** for each file submitted. The header comment must contain the following information:
  - Course number, section.
  - Instructor name.
  - Your name.
  - Name/Purpose of the program.
  - Date.

  Extra information in the header comment may be provided as well (version, extra dates - e.g., one for assignment commencement, one — for submission, etc...).

Testing and Submissions. Any submission that does not compile using the

```
gcc -ansi -Wall -Werror -lm
```

compiler settings will receive an automatic score of 0.

In this lab, you will be creating simple functions that return numeric (for the most part) values. You will be testing these functions using the testing framework introduced in Labs 1 and 2.

In this lab, you will have to write tests for your functions. Your deliverables for this lab include both the implementations of requested functions **and** C programs that test each individual function.

The Task

Note: Please consult the instructor if any of the tasks are unclear.

You will implement and test a number of mathematical functions. For this part of the assignment you will create the following files:

- `functions.h`: this file will contain implementations (function definitions) for all `C` functions you will implement.
- `test-<functionName>.c`: for each function you implement you will submit a `test-<functionName>.c` file (e.g., if you need to implement a function `getTally()`, you will submit a `test-getTally.c` file). This file will contain a `main()` function that uses the appropriate test macros (`checkit_int(X,Y)`, `checkit_double(X,Y)` or `checkit_char(X,Y)` to test one function from the `functions.h` file.

The following requirements apply to your **entire assignment**, i.e., to the implementation of every function and every `test-<functionName>.c` program.

R1. Functions to implement. The list of functions you shall implement is shown in Table 1. The table shows the function name, the return type and the types of its arguments, and the mathematical formula to compute the function.

Name	Return type	Argument types	Formula
`logBase`	`double`	`double`	$logBase(x, y) = \log_y(x) = \frac{ln(x)}{ln(y)}$
`quad`	`double`	`double`	$quad(x) = (x-1)(x-2)(x-3)(x-4)$
`eucledean`	`double`	`double`	$eucledean(x1, y1, x2, y2) = \sqrt{(x1-x2)^2 + (y1-y2)^2}$
`sim`	`double`	`double`	$sim(x1, y1, z1, x2, y2, z2) = \frac{x1 \cdot x2 + y1 \cdot y2 + z1 \cdot z2}{\sqrt{x1^2+y1^2+z1^2}\sqrt{x2^2+y2^2+z2^2}}$
`trig`	`double`	`double`	$trig(x, y, z) = \sin(xy) + 2\cos(yz) - 3\sin\left(\frac{z}{x}\right)$
`manhattan`	`double`	`double`	$manhattan(x1, y1, x2, y2) = \lvert x1 - x2\rvert + \lvert y1 - y2\rvert$
`logit`	`double`	`int`	$logit(x, y) = \log_2\left(\frac{x}{y-x}\right)$
`passerRating`	`double`	`int`	see below
$passerRating(yards, tds, comps, ints, atts) = \frac{8.4 \cdot yards + 330 \cdot tds + 100 \cdot comps - 200 \cdot ints}{atts}$			
`fn`	`int`	`int`	$fn(x, y, z) = xy + \frac{x}{z} + \frac{z+1}{x+y} - \frac{xyz}{y+1}$
`takehome`	`double`	`int`	see below
$takehome(salary, taxRate, health) = (salary - health)(1 - \frac{taxRate}{100})$			

Table 1: Functions you need to implement.

Note. `logBase()` computes the logarithm of its first argument using the base provided by the second argument. `quad()`, `trig()` and `fn()` are sythetic functions designed to test how you implement various C expressions. `eucledean(x1,y1,x2,y2)` computes Eucledean distance between points $(x1, y1)$ and $(x2, y2)$ is two dimensions. `sim()` computes a measure called *cosine similarity* between points/vectors $(x1, y1, z1)$ and $(x2, y2, z2)$ in three dimensions (essentially, the cosine of the angle between these two vectors). `manhattan()` computes the Manhattan distance between two points in two dimensions. `logit(x,y)` computes the log-odds ratio (whatever this is) of x and $y - x$. `passerRating` computes the college football passer rating based on the number of yards, touchdowns, completions, interceptions and attempts a passer (quaterback) made in a game. `takehome()` is a simplified computation of a takehome pay based on the salary, tax rate (represented as an `int` between 0 and 100) and the employee's contributions to the health plan.

R2. Function code. Each function shall be implemented in the `functions.h` file. The name of the function shall be exactly as specified in the first column of Table 1. The code for each function shall be a *single* `return` *statement.* The functions shall use no local variables.

Please note, that your implementations must satisfy the style rules. In particular, you are not allowed to have any *magic numbers* (except of 0, 1, -1 and 2) in your code. All other constants must be `#define`d in your `functions.h` file.

R3. Testing. For each function, create a file `test-<functionName>.c`. The `<functionName>` part shall be exactly as specified in the first column of Table

1. E.g., for the `logBase()` function, your test filename will be `test-logBase.c`.

Each test file shall contain just one function, `int main()`. The `main()` function, in turn, shall consist solely of the invocations of the `checkit_int()` or `checkit_double()` macros from the `checkit.h` file (as appropriate for the function). The `main()` function shall contain *at least* 10 tests you designed to test your function.

(Thus, for this lab, you will have to create a total of at least 100 tests. Note, that you will have to manually verify these tests, so this process will take significant time.)

The preprocessor directives for the test files shall include

```
#include <stdio.h>
#include "checkit.h"
#include "functions.h"
```

R4. Limitations. Your implementations of the functions described in Table 1 are responsible for **correctly computing the value of the function everywhere where the function is defined.** You are not repsonsible for the behavior of your functions on the sets of inputs where the function is not defined.

For example, `logBase(10,0)` is undefined, and therefore your implementation should not worry about the output on this pair of arguments. **All instructor's tests to all functions will only test them on sets of arguments where the function is defined**.

Grading

Your submission will be graded as follows.

Your tests. The correctness of your test cases in the test-<functionName.c> files will be verified by running your tests with the instructor's (correct) implementation of the assigned functions. Any tests that fail will be discounted and deductions shall be taken from your score.

Your functions on your tests. Your implementations of the assigned functions must pass all **correct** tests from your test-<functionName>.c file. Passing of any incorrect tests will be penalized.

Your functions on intructor's tests. Your implementations of functions will be run against the instructor's own battery of tests (about 10 tests per function, unless more tests are required to do a complete coverage of all possibilities). Your implementations are expected to pass all instructor's tests. Failing to do so will be penalized.

Submission.

Files to submit. You shall submit eleven files: `functions.h`, and the ten test-<functionName>.c

Your file names shall be as specified above (and remember that Linux is case-sensitive). We use automated grading scripts. Any submission that has to be compiled and run manually will receive a deduction.

Submission procedure. Use `handin` to submit your work. The procedure is as follows:

- `ssh` to `unix1`, `unix2`, `unix3` or `unix4`.
- Upon login, change to your Lab 3 work directory.
- Execute the `handin` command.

```
> handin dekhtyar lab03 functions.h <file1> <file2> ... <file10>
```

(note, you can submit files one by one, rather than using one command.)

Other submission comments. **Please, DO NOT submit binary files**. **Please, DO NOT submit binary files**. (this has been an issue in the past.)