Lab 4:Testing, Composition, Assignment

Due date: Wednesday, January 18, during the lab period.

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. You are continuing your acquaintance with C functions and their use.

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.

- **Short lines.** No line in your program shall be longer than 80 characters.

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

- **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.

Part 1: Testing compound expressions

To complete this task you need to follow a set of steps.

Step 1. Implementation of basic functions. Create a file `basic.h` which contains implementations of the following functions, which we will refer as *basic functions*:

1. `double f(double x, double y);`
$$f(x, y) = x + y.$$
2. `double g(double x, double y);`
$$g(x, y) = x^2 - y^2.$$
3. `double h(double x);`
$$h(x) = x^2 - \frac{x}{2}.$$

Note. For raising a value into a small integer power (square, cube), it is better to use multiplication operation than to call the `pow()` function. `C` computes `x*x*x` faster than `pow(x,3)`.

T he process of testing a single function is usually referred to as *unit-testing*. In the past two labs you've been mostly unit-testing individual functions that you created. On this step you are asked to unit-test the basic functions you implemented in **Step 1**.

Create a `C` program `test-basic.c`, which contains only one function, `int main()`. The `main()` function shall consist of a sequence of invocations of the `checkit_double()` macro. You shall create five test cases for each basic function for a total of 15 tests. The order of tests is: tests for `f()`, then tests for `g()`, then tests for `h()`.

All tests shall be successful.

Step 3. Testing functional compositions. Create three `C` programs with file names `composition01.c`, `composition02.c` and `composition03.c`.

Each file shall consist of a single `int main()` function. Each function shall contain 10 `checkit_double()` test cases testing one of three *composite expressions* that use basic functions created by you on **Step 1**.

The following composite expressions shall be tested.

- `composition01.c` shall contain test cases for functional composition

  ```
  f(g(x,y),h(y))
  ```

- `composition02.c` shall contain test cases for functional composition

  ```
  g(x, f(h(x),x))
  ```

- `composition02.c` shall contain test cases for functional composition

  ```
  h(g(x,f(x,g(y,x))))
  ```

Note: To properly devise test cases for these expressions you need to deconstruct each expression and simplify them. For example if you have an expression `f(f(x,y),x)` you can represent it as follows:

$$f(f(x,y),x) = f(x,y) + x = x + y + x = 2x + y.$$

Once you simplified the expression (on paper), you can devise test cases for it, and confirm, through the actual testing process the validity of these tests.

Submission notes. For this part of the assignment, you will submit five files: `basic.h`, `test-basic.c`, `composition01.c`, `composition02.c` and `composition03.c`.

Part 2: Use of assignment.

In this part of the lab you start incorporating the use of variables and assignment statements in your arsenal.

For this assignment you will implement a collection of functions, and write simple programs that use the functions you have imlpemented. Your functions will use local variables and assignment statements (as well as `printf()` calls to produce results.

Function library. All functions you implement for this part of the lab shall reside in a file named `mylib.h`. The list of functions to implement and the necessarys specifications are provided below.

Time conversion function. You shall implement a function

```
void convertTime(int seconds);
```

which takes as input an elapsed time value represented in seconds, converts this value into elapsed time represented in terms of *days, hours, minutes* and *seconds*. The function shall end with a `printf()` statement that outputs the collected information using exactly the format specified below:

```
Elapsed time: xxx days, yyy hours, zzz minutes, www seconds
```

where `xxx`, `yyy`, `zzz` and `www` are replaced with computed values for total elapsed days, hours, minutes and seconds respectively. The output should end in a new line.S

Example. The call `convertTime(90330)` shall produce the output

```
Elapsed time: 1 days, 1 hours, 5 minutes, 30 seconds
```

(i.e., 90330 seconds equals to one day (24-hour period), one hour, five minutes and 30 seconds).

Note: It is ok for you to output `days`, `hours` etc. in plural, even if only one day or one hour, etc are reported.

You are responsible for correct behavior of the function on non-negative inputs. Negative inputs will not be considered.

Text encoding function. Implement a simple encoding function that encodes 3-letter words and prints out the original word and the encoding. The function declaration is:

```
void encode3letters(char c1, char c2, char c3);
```

Input letters are *lowercase Latin alphabet letters.* (You are not responsible for behavior of the function on other inputs).

To encode each letter we use a special variant of the shift encoding discussed in an earlier lab:

1. the first letter of the word is encoded with the shift of 4;
2. the second letter of the word is encoded with the shift of 16;
3. the third letter of the word is encoded with the shift equal to the distance in the alphabet between letter `'a'` and the first original letter of the word.

The function shall end with a single `printf()` call which produces the following output

```
XYZ --> ABC
```

where `X`, `Y` and `Z` are replaced with the first, second and third input characters respectively, while `A`, `B`, `C` are replaced with the computed encodings of the first, second and third characters respectively. (the printed message shall end with a newline).

Example. Consider the function call `encode3letters('d','o','g')`. The encoding will proceed as follows.

1. The first character, `'d'`, is shifted by 4 positions: `'d'+4` is `'h'`.
2. The second character, `'o'`, is shifted by 16 positions. `'o'` is the 14th letter. Shift by 16 positions is wrapped around ($15+16 = 31 \bmod 26 = 5$) and therefore, the encoding of `'o'` is `'e'`.
3. Finally, the third character is `'g'`. Its shift is the distance between the first character, `'d'`, and the character `'a'`. `'d' - 'a' = 3`, and therefore, `'g'` is shifted by three positions and becomes encoded by `'j'`.

Therefore, the function will print the following text

```
dog --> hej
```

Statistics computation function. Write a function, that takes as input four numbers and computes and prints out a number of statistical values. The function declaration is

```
void stats(double a, double b, double c, double d);
```

The function shall compute the following statistical values:

1. AV: the average of all four inputs:
$$AV = \frac{a+b+c+d}{4}.$$
2. $STDEV$: the standard deviation of the distribution of the four inputs:
$$STDEV = \sqrt{\frac{1}{4}\left((a-AV)^2 + (b-AV)^2 + (c-AV)^2 + (d-AV)^2\right)}.$$

The function shall end with two `printf()` calls which print the values of AV, $STDEV$ and $Pearson$ in the following form:

```
Average = XXXX
Standard Deviation = YYYY
```

Here, `XXXX` and `YYYY` shall be replaced with the computed values of AV and $STDEV$ respectively.

Example. The following call `stats(1,2,3,4)` shall result in the following output:

```
Average = 2.500000
Standard Deviation = 1.118034
```

Implementation note. While you can certainly implement all computations inside `stats()` function, a better way to do it is to declare (and define) two more C functions: one for computing the average of four numbers and the other — for computing the standard deviation of the four numbers. These functions can be debugged using our `checkit_double()` framework. You can then simply call them from the `stats()` function.

Submission notes. While you are fully expected to create test programs that verify the work of the three functions you were asked to implement in this section, you only need to submit the `mylib.h` file for this part of the assignment.

Submission.

Files to submit. You shall submit six files:`basic.h`, `test-basic.c`, `composition01.c`, `composition02.c`, `composition03.c` and `mylib.h`.

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 lab04  <files>
```

(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.)