Programming Assignment Protocol

For all programming assignments in this course, please follow the protocol specified in this document unless otherwise directed.

Process Document To Submit

You must submit a document that chronicles your work throughout the assignment, which will be referred to as the "Process Document". The process document should be .doc, .docx or .pdf file and be typed. It should have the following sections:

Planning Phase

In this section discuss the data structures you were planning on using (arrays, structs, etc.) as well as other planning you did, such as what functions you might want to have. This doesn't have to match your final design, but just show that you thought about it. What you may want to do is after the fact, explain why a certain plan of yours didn't work (this frequently happens), and add it onto this section.

Assistance Received

In this section outline the assistance you received when you got stuck working on the assignment. This may include discussions/emails with the course staff, looking up information online (how do I use strlen?) or minor help you obtained from another person while debugging.

Debugging Phase

In this section discuss the errors you made and how you caught them.

Testing Phase

In this section, discuss test cases you created on your own to test your program AFTER your program correctly solved the sample cases.

The goal of the process document is to prove that you did your own work for the entirety of the project.

Please name this document LASTNAME_PROCESS.doc (or .docx or .pdf).

This should be submitted for the week 1 programming assignment, the five individual programming assignments and the 2 Kattis programs.

Code File to Submit

Each program will require a submission one or more .c files that solves the problem or problems at hand. Please name your files the requested file name.

Test Case Format, use of stdin, stdout

Most of the assignments will follow a strict format where your program will have to read through several test cases. Your program should read its input from standard input (so use scanf) and print all output to standard output (so use printf). You should NOT try to store all information for all test cases simultaneously. Rather, you should declare variables INSIDE your case loop to store information to solve one test case. Here is a mold of what most of your programs will look like:

```
#include <stdio.h>

// Other stuff before main.

int main(void) {

    int numCases;
    scanf("%d", &numCases);

    for (int loop=0; loop<numCases; loop++) {

        // Declare variables to store data here.

        // Call functions as necessary to solve problem.

        // Output solution to test case.

    }

    return 0;
}
```

How to Test Programs on your Own

To test a program on your own, please type up some test cases in a file first and give that file a name. (I typically call my files problemname.in, where problemname is the name of the problem or the name of the .c file to be submitted for the assignment.) Place this file in the same directory as your .c and .exe files.

Then, after compiling your program, open up Command Prompt (in Windows) or Terminal (Mac). Change directories to where both the executable and test input file are. (cd is the command to change directories.)

Then, on the command line, do the following:

```
>programname.exe < testfile.in
```

When you do this, the output should come in the terminal window. If you want the output to be written in a file instead, do the following:

```
>programname.exe < testfile.in > myanswers.out
```

Now, when you do this, there will be no output to the screen as it's been redirected to the file myanswers.out. When you do this, make sure no meaningful file named myanswers.out exists. If it does, this will overwrite those contents.

Now, if you know what the correct answers are and store them in testfile.out, you can see if your answers match by doing this on the command line:

```
>fc myanswers.out testfile.out
```

If the files have the same exact contents, then this will produce the result:

```
fc: No differences found.
```

or something to that effect.

If they don't fc produces some output that is hard to follow. In these cases, you may want to use a tool like WinMerge, which does a better job of highlighting where two files are different. You can download WinMerge here:

https://winmerge.org/downloads/?lang=en

This will be covered in detail in the sample assignment video.