# COP 3502 Programming Assignment Protocol - Fall 2021

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

## Testing Documents To Submit

You must submit a document that shows that you thought about testing your program beyond the sample data provided. In the real software cycle, testing and debugging represent a majority of the work (more than 50%) done, yet in most courses, no grade is given for this portion of the process at all. In an effort to make sure students spend time thinking about how to test their programs, a portion of each individual program grade (roughly 15%) will come from the testing document and files you submit.

The document should ONLY discuss test cases you created NOT INCLUDING the sample provided.

You will submit 3 files in all for this portion of the grade:

```
LASTNAME_TESTING.doc (or .docx or .pdf)
progname.in
progname.out
```

The first document discusses in English, what types of cases you felt were good to create to verify the correctness of your program. Discuss how you designed the cases and how you calculated the correct answers to them, independent of your program. (Note: in some cases, it's valid to use an easier brute force program that runs slower to verify answers.) **If you generated some of your test cases with a program, discuss at a high level how you generated the cases.**

The last two files should contain both the test cases you created (either typed by hand or generated via a program.) and the correct solutions to those test cases.

The goal of the testing document is to make students how to create their own test cases (very valuable skill) and to hopefully help students catch errors they may not otherwise, which, in turn, ought to improve student grades.

This document should be submitted for all individual programming assignments.

## 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.