# COP 3502 Programming Assignment Protocol - Fall 2023

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

For each program, you will be asked to submit **one or more source files (.c)** containing a program(s) to solve the given problem(s).

## Input Format

We will test your program on several test cases. Your program, however, should be written to process only **one test case** from **standard input**, writing the output to **standard output.** (This means you read in input using either scanf or getline, and you will produce output via the printf function.)

When your program runs, **it should NOT prompt the user to enter any information**. Rather, when we test your programs, we pipe input into standard input by running the program from the command line with file redirection.

Each assignment will come with an "Input Format" description. *It is guaranteed that the input will conform to the specification given*. Thus, if it says something like, "$0 < \boldsymbol{n} \leq 10$", then in ALL of our input test files, ***n*** will be an integer in between 1 and 10, inclusive. Thus, **do NOT do:**

```
if (n < 1 || n > 10)
    printf("This is not a valid test case.\n")
```

in your code.

## Testing Your Program

In the problem descriptions of your assignments, you will be given "Sample Input" and "Sample Output." **THESE DO NOT REPRESENT COMPREHENSIVE TEST CASES.** In fact, just because your program produces the correct output on these cases doesn't mean that your program is actually correct. These test cases are provided as a courtesy, to make sure students understand the input format. They tend to be very easy cases, ones where many incorrect algorithms arrive at the correct answer anyway.

Thus, it's critical that you **create your own test cases** to improve your confidence that your solution is correct. In past semesters I required students to create a minimum of 5 test cases along with a document describing what they were testing and why. I've decided not to require this document this semester, mostly to remove the extra grading burden from my teaching assistants. This does not mean you should not create your own test cases. In fact, I can guarantee that doing so will likely help you catch errors, which, in turn will allow you to fix those errors, resulting in an improved grade.

Thus, I have decided to treat you like adults and not children. I am telling you what a good practice is but not requiring it, letting you decide on your own what to do. The consequence of this is that some students won't take this extra step and for those students, there's a good probability that skipping that step will lower their grade on the homework assignment. My hope is that this happens

and students learn a lesson and then start testing =) What would be even better is that students trust my 24 years of experience teaching and just go ahead and thoroughly test their programs.

Here is how I think students should spend their time on assignments:

20% reading the assignment carefully making sure that students know the skills that it's testing. If not carefully review those skills by writing some small practice programs.

10% planning a solution to the assignment on paper.

20% initially coding a solution to the assignment.

20% writing extra test cases beyond the sample to test the assignment

30% using the results from the test cases to debug and fix the program

Generally speaking, in industry a vast majority of time is spent on testing and debugging; very little is actually spent on developing new code. The better the planning phase is, the less debugging must occur, thereby reducing the overall time of the testing and debugging phase and reducing the total amount of time spent on the program.

**The biggest mistake students make in this class is that they start writing code before they understand what they need to understand to solve the given problem. It's my experience that when students do this, they spend 15-20 hours on an assignment and never solve it correctly. Had those students spent 5 hours planning and making sure they understand the tested concepts, they would complete a correct program in the next 5 hours, greatly reducing the total time spent.**

**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 process one test case. Your program should read its input from standard input (so use scanf or getline) and print all output to standard output (so use printf). Here is a mold of what most of your programs will look like:

```
#include <stdio.h>

// Other stuff before main.

int main(void) {

    // 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.) Note that in some environments, you may have to add "./" before the name of the executable program. (That's a dot and forward slash…)

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.