How to test your code?

Software engineers constantly test their code for bugs and edge cases, semantic and syntactic errors to make sure their program is ready for production. Testing your code means running the code with a controlled input and checking for the intended output. Here are some practices we strongly suggest you follow to meet CSCI 335 assignment requirements.

1. **Understand the prompt to make sure you provide the correct input scenarios to correctly and thoroughly test your code.** In other words, CAREFULLY read the assignment requirements document end-to-end and plan your solution approach. Failure to do so can result in wasted hours with a solution that either doesn't work, or solves a different problem. You should not be jumping straight into coding - think for a bit about how you would solve it.

2. **Write self explanatory code** in a way that anyone who reads your program can pick it up easily and understand what's going on. This is also extremely helpful when you review your own code at a later time.
   a. **Always** add a brief description of what a function does before the function declaration. **Documentation is a required part of your assignment submission**.
   b. *If needed: add comments prior to variables.
      i. Do not describe **what** something does, but rather **why** it's needed.
      ii. Do not over comment your code because you could end up confusing the reader.
      iii. Only write what is necessary to understand the code.

   (This will also allow TA's and graders to understand your code and help debug your code better!)

3. **Test your code in increments** to find small bugs earlier instead of trying to fix big bugs later. It's easier to debug programs with less code.

4. **Consider edge cases** to avoid unexpected outputs and your program crashing.

5. **Create and verify test cases** to potentially find more edge cases and to ensure that your program runs as expected. This will be critical for your assignments since there will be both known and unknown test cases as discussed in class.

6. **Review** the document by Dr. Ligorio about ensuring your submission works on Gradescope as expected: Gradescope Help .

Consider the following prompt. "Given an array of integers, find the average value of that list". We now provide a couple of examples to motivate how you would go about testing the code.

1. **Understand the prompt**: You are given an array of integers as input, and you want to find the average. Make sure you don't confuse "average" with "median".
   a. Average = the sum of all values divided by the input size.
   ~~b. Median = the middle value of a sorted dataset~~ //not what you need

2. **Write self explanatory code**: Suppose you approached this by computing the total sum, then dividing by input size.
   a. You would want an integer variable to store the sum of all values. Do not name it "x", "bob", "jacobson", etc. That would not be very helpful.
   b. Give it a meaningful name along the lines of: "sum", "totalSum", etc.
   c. Do not name it something unnecessarily long like "sum_of_all_int_values_in_given_array" or "sum_of_all_int_values_in_given_array_that_I_might_eat_for_breakfast".
   There are rare exceptions for naming length, but you get the point.

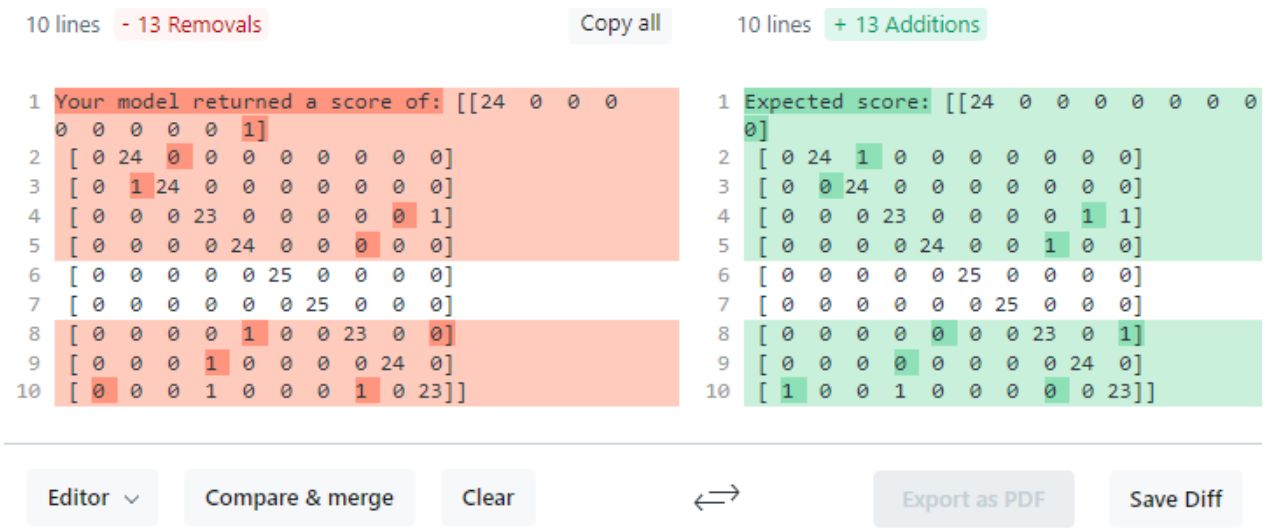
3. **Testing in increments**: You can split this prompt into two parts:
   a. Find sum of input values //debug and ensure output is correct
   b. Divide by input size //debug and ensure output is correct
   With this approach, you will implement sum first, then divide. If you know that the sum works perfectly fine but the division does not, then you will know which section of your code to debug.

4. **Cover Edge Cases**: Even with a straight forward prompt like this, edge cases occur in various forms. Consider the following:
   a. Most of the time, the average won't be a whole number. So you will want to store sum as a <double> type.
   b. If the input size is empty, then you might accidentally divide by zero. In such a case, you would just return 0.
   c. What if you meet INT_OVERFLOW on summation? If that happens, then you need to refactor your algorithm to handle such cases.
   d. How does this algorithm scale? Can you improve the space/time complexity?
      i. This algorithm will take O(n) time and O(1) space. Looks pretty good, and doesn't seem like there's much room for improvement.
      ii. In the case where the time complexity is exponential or some high order polynomial, please consider refactoring the algorithm (exceptions apply).
   e. It is very likely that you will miss some edge cases, so you should make a bunch of test cases to make sure your program works.

5. **Creating Test Cases**: Generate a bunch of cases - small enough that you can solve by hand, yet large enough that it can be used as a guideline for the expected behavior of your program. As you test your program on these cases, it's likely to come across some issues.
   a. For this prompt, you can just compute the average using an online calculator and matching the output of your program with the expected output. However, for less simple prompts, you definitely want to consider doing about 5-6 cases by hand, and adding in 2-3 edge cases.
   b. Test cases vary based on the problem, but keep some of these in mind:
      i. Strictly increasing/ decreasing
      ii. Random order
      iii. List containing 4 or less elements
      iv. Varying length inputs
      v. Contains Duplicates
      vi. Etc.
   c. If needed, you can try creating a program that generates test cases for you.

Regarding Gradescope:
Now that you have completed your function and see that it works and covers a variety of test cases correctly, are you Ready for submission?
First before submitting to Gradescope, you MUST ensure that your program compiles (with g++ using C++14) and runs correctly on one of the Linux machines in 1001B lab at Hunter North. The lab machine is your baseline. Additionally, make sure your program works as intended per the assignment specification. Although Gradescope allows multiple submissions, it is not a platform for testing and/ or debugging and it should not be used for that. You **MUST** test and debug your program on your local machine.

Some things to note:
- Gradescope will not read your program's main function.
- Depending on how gradescope provides feedback, you can also try out https://www.diffchecker.com/ to check differences between texts



Resources you can check out:

- [Unit Testing Article by techtarget]
- [GoogleTest Documentation]
- [How to Write Test Cases - testlodge]