

This lab has **five (5)** checkpoints.

This lab is due at the **end of the lab period**. Lab checkpoints will not be accepted after that.

Learning Outcomes

Upon completing this assignment, students should be able to

- Read a programming assignment and *outline* application level tests for the solution.
- Read a README and corresponding program, *run* the program, verify the test plan from the documentation.
- Use their own test plan to test a solution for a given assignment.
- Write up a test report for a program.

Introduction

Read the provided assignment, `data/p001Adder.pdf`, *carefully*. Instead of programming a solution for it, you will *test* and *evaluate* some solutions for it.

You will **not** see the Java source code for the solutions you are testing. The tests will, therefore, be *blackbox* tests where you can only control the *input* to the application and observe the *output* from the application. You will select your input, use the assignment to determine what the *expected* output should be, and, only then, run the program with the given input, noting the *actual* output. You will note and explain anytime the actual behavior varies from the required behavior in the assignment.

About Tests. Edsger Dijkstra famously said, “Testing can only prove the presence of bugs, not their absence.” And Michael Bolton (no, not *that* one) said, “A good test is one that has a high probability of breaking the code if there is a problem.”

The point of this comment is that a good test is designed to *break* things unless the code is correct. Look for edge conditions that the programmer should have tested for, technically correct¹ input that the programmer may not have thought about, or errors that users might make. Learn to do this with your own code and you will soon plan ahead and fail a lot fewer tests. And know that the bugs you tested for are not there, at least not in the way you thought they would be.

Method

1. Read the provided **Adder** assignment (`data/p001Adder.pdf`). The *requirements* describe what a solution **must/must not** do. The *sample interactions* give a set of input/output pairs to illustrate correct responses. The assignment also lists all *errors* that the solution must handle and what the correct **response** is.

Summarize the assignment (using the outline structure below) *as if* you were about to start writing a solution. Be sure to note how the user provides input. Do not obsess on where any given note goes but try to capture everything that a solution should do.

- I. Requirements
- ...
- II. Sample Input/Output Pairs
- ...
- III. Errors and How to Handle Them
- ...
- IV. Other Observations
- ...

¹The **best** kind of correct.

✓ Show your written, *legible* outline to the lab instructor. Be prepared to explain where any given part of your outline comes from in the assignment.

2. Draw a *test plan*, a table with three columns for the input, expected results, and the actual results. Add a row to the table for each *test case* in your outline, leaving the last column blank because you have not yet tested anything.

Example: the Adder assignment says that the empty command line is to be treated as summing zero floating point values. That requirement gives the following test case in the table:

Input	Expected	Actual
<i>none</i>	0 values added; sum = 0.00	

✓ Show your complete test plan to the lab instructor. Be prepared to explain any give test case *and* why you believe you have all necessary test cases.

3. **AdderA** Student A, call him Aloysius ², has turned in the solution in the data/AdderA folder. Apply your test plan to his solution.

Using your test plan, test the solution. Run each test case and make a note of the *actual* results.

As you run test cases, new test cases may occur to you. Add them to the end of your test plan; make a note of *why* the new test case came to mind (this helps the next time you come up with tests).

✓ Show your completed test plan to the lab instructor. Be prepared to talk about the correctness (or, perhaps, lack of correctness) of AdderA. Expect to have to explain why your test plan tests all of the requirements in the Adder assignment.

4. **AdderB** Brunhild ³, turned in the solution in the data/AdderB folder. Apply your test plan *again* to test her solution. This means starting over with the first two columns as before and filling in a *new Actual Output* column.

✓ Show your completed test plan to the lab instructor. Be prepared to talk about the correctness (or, perhaps, lack of correctness) of AdderB, justifying the completeness of your results.

5. AdderB behaves oddly when given a single input value. Looking at your results above, can you explain what the result *should* be when there is a

- single *integer* value
- single *floating-point* value
- single *mistaken* value (one that cannot be treated as a number)

Finish each of the above **Expected** results with the **Actual** results for AdderB. Do these same errors plague longer or shorter input?

What do you believe is happening *inside* Brunhild's program that causes these errors — or, is this actually *one* error that just looks different with varying input? **Write** a short description of what you think is happening.

✓ Show your three expected/actual pairs and your speculation to the lab instructor. Be ready to *explain* how your error[s] would cause the problems you observe.

²From Germanic: "famous warrior".

³From Norse: "armored protector of battle"; Valkyrie queen.