CS 1632 - DELIVERABLE 6: Testing Strategy for RPN++

By: Joseph Reidell(jreid2f) & Mohit Jain(zaxway)
URL: https://github.com/jreid2f/CS1632-Deliverable6

Program's Overall Quality

File Check - All checks seem to run as normal. (Green)

Error Display - Error messages display as normal. (Green)

Keyword Check - No non-trivial issues. (Green)

Expression Creation - Expressions are created as normal. (Green)

Display Prompt - No non-trivial issues. (Green)

Input Result - Results from expressions are shown as normal. (Green)

File Result - Results from files are shown as normal. (Green)

REPL - No non-trivial issues. (Green)

RPN - No non-trivial issues. (Green)

Areas of Concern

Currently, no defects have been found in the program.

Rubocop does not display any offenses at all for any of the 5 class files.

In addition, the test coverage for the files exceeds the required 70%.

Simplecov Percentage: 87.37% covered.

Testing Strategies

In order to fully test this program, our team had to make use of a wide variety of assertions per test. For most test cases, we had to test to make sure the program could support extremely large numbers, and that regardless of the input that we entered, the program would never display any inconsistency. We had to make that the errors for each “bad value” test were accurately displaying and that the errors were not getting mixed around. In addition, our primary reason for testing extremely large numbers and invalid input relied on our primary goal that regardless of what we entered, the program would never crash. Some of the tests we faced a number of the issues with were tests that required the use of files in order to properly run. In addition, one other property of testing that made it really difficult and tedious was the fact that a lot of methods greatly relied on one another in order to function. We noticed cases of methods that would call another method, and then that method would call another method, and this pattern would continue on throughout; therefore, in order to forge the test cases, careful study of the program was required. To answer the question of how much time we devoted to each kind of test, it really depended on what I was testing. Some methods required us to test multiple different cases. Let’s say, for example, one of the methods we tested involved testing valid and invalid inputs. We would have to test multiple test cases in this method then. We would have to test a set of valid inputs, normally involving up to 6 valid inputs (2 inputs being normal test cases, 4 inputs being an edge case), and 3 invalid inputs (2 inputs being normal test cases, 1 input being an edge case). Tests like these would take a little longer to fully cover than other tests which are more specific. Most of the tests I wrote however were unit tests. I did not feel the need to use property tests because there is no guarantee that if I use a property test, it will test the values I had in mind to test. In addition, property tests and testing with files are not a good combination. I mainly used unit tests because I felt that I had more freedom to test exactly the values I wanted to test in the program, and I felt that my range of values would give an excellent coverage for each test.