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

By Nigel Bernard & Josh Koshy

GitHub: https://github.com/nib58/DELIVERABLE6

**Arithmetic Operations:** The Arithmetic Operations have been fully-operating. We utilized the arithmetic functions under the java package “BigInteger” to replace the normal integer variables in order to facilitate integer overflow. All operations are working functionally for any integer. All errors with arithmetic operations have been covered.

**LET command:** The LET command has been fully-operating. We used a “BigInteger” array instead of a normal integer array. Using the “LET” command only uses a small amount of performance on top of normal arithmetic operations. All errors with LET and variables have been covered.

**PRINT command:** The PRINT command has been fully-operating. The PRINT utilizes very little in performance. All errors with PRINT have been covered.

QUIT command: The QUIT command has been fully-operating. The QUIT command utilizes very little in performance. All errors with QUIT have been covered.

**Integer Overflow Handling:** The java package, “BigInteger”, was utilized to solve the handling of very big integers (greater than 2,147,483,647). All arithmetic operations stem from the methods under BigInteger and have been working functionally. All errors with Integer Overflow have been covered.

**File Handling:** The program goes through each file in order, line by line, and acts as if all the files are as one. Everything is working functionally with handling files. REPL mode has been sure to be turned off and all differences are accounted for. All errors with file handling have been covered.

**REPL Mode:** REPL mode is determined to be turned on when there is no argument(s) prompted when initially running the program. All differences between the mode being off and on have been accounted for. All errors with REPL mode have been covered.

**System Performance:** Overall, the performance is running well. There could be more improvement found in refactoring the method dealing with the arithmetic, but we are not too sure how much exactly. Any improvement could be negligible, but we won’t know until we put effort into analyzing it further.

**User Interface:** All requirements with the look of the program has been accounted for. All input and output should look as expected.

Areas of Concern

None to be found

Testing Strategies

We decided to use a pyramid testing strategy (70% Unit Tests, 20% System Tests, 10% UI Tests).

We ran automated tests on our Unit Tests and System Testing and manual tests on our UI Tests because we felt that it would be more time efficient and practical in that way. Our UI Tests consisted of Performance testing of the program from using a VMProfiler, and a manual test of all aspects of the program (declaring a variable, printing, writing to a file and quiting). The most time and effort was dedicated to the unit tests. By focusing on unit tests, we made sure that our program first was proficient in its happy path, then used manual testing to check form the more obscure corner cases. To help increase the quality of our program

The parts of the code that need to be cleaned up

