Christian Flickinger, Bin Dong

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

<https://git>hub.com/cef39/reverse-polish-notation

Red-Yellow-Green template

GREEN: REPL mode correctly loops, takes in user input and responds accordingly.

GREEN: All required keywords are parsed and run correctly.

GREEN: User able to enter filenames with RPN commands and program parses and returns correct values. Program handles the case of multiple files.

GREEN: Integer overflow taken care of through the use of the BigInteger class.

GREEN: Program correctly performs arithmetic operations given RPN input.

GREEN: Program handles most, if not all, cases of bad user input and does not crash.

GREEN: Most, if not all, of the error messages accurately report what the cause of the error is.

Areas of Concern

When invalid RPN expression (such as ‘LET a 1 2’) is applied to LET command, program displays error of invalid RPN expression (such as ‘2 elements in stack after evaluation.’) instead of error of the LET command. The variable however does not get initialized and requirement 19 does not state what the error message should be in this case, so this is what we went with.

No known defects.

Testing Strategies

Unit tests were primarily used for testing the code. Using these unit tests, we were able to verify that the actual evaluation of RPN was correct. We tried to cover all sorts of edge cases such as division by 0, or integer overflow. Through some of our unit tests, we discovered that a long wouldn’t be enough to cover integer overflow and had to replace everything with the BigInteger class in order to cover that requirement.

We also performed a good amount of UI testing in order to test all the possible errors that may arise from user input, and were able to write try..catch blocks in order to cover all of those cases. We used the examples provided in the deliverable6.md as a base for this testing.

Writing the unit tests took most of our time due to how long it can take to write code, and because we wanted to thoroughly test all different possibilities. UI testing didn’t take nearly as long but was just as important since we wanted to make sure we displayed the correct errors to the user.

One of the biggest recommendations we have is to test while you write the program. It’s nice to be confident that the method you wrote is correct and that you can continue onto the next part of the program.

The evaluateRPN() method could be improved upon. As of right now, it has about 70 lines of code and has tons of IF statements and try..catch blocks. We could break this method into multiple methods so that each can be easily unit tested and it would make the program more readable.

We discovered the importance of a thorough test plan throughout this class/project. When beforehand, one would have to hope that their project worked completely, but with a good test plan, one can be more assured in the accuracy of their program.

Manual Testing

All of our manual testing consisted of creating all of the files listed on the requirements6.md and running them to match our output with Laboon’s output. We did the same thing with the sample REPL output to make sure we got the same output.

ID: REPL Test 1

Summary: Testing basic REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “4 3 +”

Postconditions: 7 should be result

ID: REPL Test 2

Summary: Testing basic REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “LET a 10”

Postconditions: 10 should be result

ID: REPL Test 3

Summary: Testing basic REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “a 1 +”

Postconditions: 11 should be result

ID: REPL Test 4

Summary: Testing basic REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “2”

Postconditions: 2 should be result

ID: REPL Test 5

Summary: Testing basic REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “LET b 2 2 +”

Postconditions: 4 should be result

ID: REPL Test 6

Summary: Testing basic REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “let c a b +”

Postconditions: 14 should be result

ID: REPL Test 7

Summary: Testing basic REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “print c”

Postconditions: 14 should be result

ID: REPL Test 8

Summary: Testing basic REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “print c 1 +”

Postconditions: 15 should be result

ID: REPL Test 9

Summary: Testing basic REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “quit”

Postconditions: program should exit

ID: REPL Invalid Test 1

Summary: Testing invalid REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “a 4 +”

Postconditions: Error should be ‘Variable a is not initialized’

ID: REPL Invalid Test 2

Summary: Testing invalid REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “1 2 + +”

Postconditions: Error should be ‘Operate + applied to empty stack’

ID: REPL Invalid Test 3

Summary: Testing invalid REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “3 4 5”

Postconditions: Error should be ‘3 elements in stack after evaluation’

ID: REPL Invalid Test 4

Summary: Testing invalid REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “LOOP”

Postconditions: Error should be ‘Unknown keyword LOOP’

ID: REPL Invalid Test 5

Summary: Testing invalid REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “4 3 LET + a”

Postconditions: Error should be ‘Could not evaluate expression’

ID: REPL Invalid Test 6

Summary: Testing invalid REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “LET a”

Postconditions: Error should be ‘Operate LET applied to empty stack’

ID: Big numbers test 1

Summary: Testing Big numbers REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “999999999999999999 999999999999999999 \*”

Postconditions: result should be 999999999999999998000000000000000001

ID: Big numbers test 2

Summary: Testing Big numbers REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “LET a 0 999999999999999999999999999 -”

Postconditions: result should be -999999999999999999999999999

ID: Big numbers test 2

Summary: Testing Big numbers REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter “LET a 0 999999999999999999999999999 -”

Postconditions: result should be -999999999999999999999999999

ID: Big numbers test 3

Summary: Testing Big numbers REPL command

Preconditions: run ‘java RPN’ in command line

Execution: Enter

LET a 0 999999999999999999999999999 –

Let b -1

a b +

Postconditions: result should be -1000000000000000000000000000

ID: RPN with files test 1

Summary: Testing reading in RPN riles

Preconditions: run ‘java RPN File1.rpn’ in command line with File1.rpn containing the RPN lines listed on Laboon’s Github for File1.rpn

Execution:

Postconditions: result should be 3

ID: RPN with files test 2

Summary: Testing reading in RPN riles

Preconditions: run ‘java RPN File2.rpn’ in command line with File2.rpn containing the RPN lines listed on Laboon’s Github for File2.rpn

Execution:

Postconditions: result should be

0

0

0

ID: RPN with files test 3

Summary: Testing reading in RPN riles

Preconditions: run ‘java RPN File1.rpn File2.rpn’ in command line

Execution:

Postconditions: result should be

3

0

0

0

ID: RPN with files test 4

Summary: Testing reading in RPN riles

Preconditions: run ‘java RPN File2.rpn File1.rpn’ in command line

Execution:

Postconditions: result should be

0

0

0

ID: RPN with bad files test 1

Summary: Testing reading in bad RPN riles

Preconditions: run ‘java RPN Bad.rpn’ in command line with Bad.rpn containing the RPN lines listed on Laboon’s Github for Bad.rpn

Execution:

Postconditions: result should be ‘Line 3: 3 elements in stack after evaluation’

ID: RPN with bad files test 2

Summary: Testing reading in bad RPN riles

Preconditions: run ‘java RPN Bad2.rpn’ in command line with Bad2.rpn containing the RPN lines listed on Laboon’s Github for Bad2.rpn

Execution:

Postconditions: result should be ‘Line 1: Unknown keyword QUOMBLE’

ID: RPN with bad files test 3

Summary: Testing reading in bad RPN riles

Preconditions: run ‘java RPN Bad3.rpn’ in command line with Bad3.rpn containing the RPN lines listed on Laboon’s Github for Bad3.rpn

Execution:

Postconditions: result should be ‘Line 2: Operator + applied to empty stack’

ID: RPN with bad files test 4

Summary: Testing reading in bad RPN riles

Preconditions: run ‘java RPN Bad4.rpn’ in command line with Bad4.rpn containing the RPN lines listed on Laboon’s Github for Bad4.rpn

Execution:

Postconditions: result should be ‘Line 3: Operator + applied to empty stack’