Joshua Kephart

GitHub username: jak243

Repository: https://github.com/jak243/1632D6.git

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

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

**Red-Yellow-Green Quality Review**

REPL Mode No issues found, could make code 5 errors more explicit

File interpreting No issues found, could make code 5 errors more explicit

RPN Class Methods are a little verbose or redundant, but functional

String Class Modifications All new methods accomplish their goals

Handling Reverse Polish Notation Accurately handled

RPN Error Class All errors functional

Execution Time No work has been done to measure or improve execution time. Work in this regard is completely stalled.

**Areas of Concern**

I have not found any defects. Though I feel that I have not been adequately able to test a lot of the situations not covered by the unit tests.

I did find 5 different types(15 total) RuboCop offenses in rpnclasses.rb (none in rpn.rb):

Class RPN has too many lines [132/100]:This is heavily linked to the verbose nature of the methods.

Multiple methods in RPN have too many lines(process, process\_repl, what\_type, arithmetic, operate, let): By far my most pervasive offense, but I could not think how to make them more concise without splitting them up further in what felt like adding unnecessary modularity

Multiple methods have high cyclomatic complexity (what\_type, operate):

Multiple methods have high Assignment Branch Condition size (what\_type, arithmetic, left):

Multiple methods have high perceived complexity (what\_type, operate):

In general all these rubocop offenses come from methods being too long and doing too much. But for the purposes of this assignment I felt like it was a very natural way to distribute the work. Though I realize that is probably because I am still a java minded man.

Lastly, as indicated in the Red-Yellow-Green review, the biggest flaw with this program would be its runtime (unless I have missed another error). This is because I did not adequately plan out my testing strategy and time management to include runtime testing or research. Because of this I have done no additional runtime optimization, and it is unlikely that it is close just with my natural implementation.

**Testing Strategies**

1. I exclusively wrote Minitest unit tests. These were the backbone of my testing strategy followed closely by manual testing I did (without a plan). I tried to cover each method well and then check the surface level things using manual testing. I found the majority of the defects (I fixed all the ones that I found) through my unit tests. I also used rubocop to do some linting.
2. I put about 80% of my testing effort into the unit tests. 15% into the relatively haphazard manual testing. And 5% into Rubocop
3. I discovered that initially my string method letter? Returned true if the string contained a letter, even if it wasn’t one letter. I also found that I had multiple problems with storing the variables and the values. There was an inconsistency where sometimes I was storing the value as a string and sometimes as an integer. I discovered that typing ctrl+ c would end the program with an exeption when in repl mode. I discovered that at first, when in repl mode, a user entering no input would result in an extra newline character being printed to stdout.
4. I did not develop manual testing plan, though I did follow the sample outputs and made sure to try to reproduce all raised errors.