# Testing Strategy for RPN+

Author: Neil Patel
GitHub Username: neilpatel

# Summary

**Program Description:**

Our program is a Reverse Polish Notation calculator that can be run in two different modes. The first mode is file mode, which is run with an additional file argument. The second mode is REPL mode (read-eval-print loop) which is run when there is no additional argument. For both cases, an input string is parsed as an array of elements that is then evaluated as a representation of specific types for program operations. These types are represented as a number and correspond as an: Integer (1), Variable (2), Operator (3), Keyword (4), Unknown Keyword (5) or Invalid Token (0). These type representations are then matched with an appropriate regex, which then calls the appropriate method needed for RPN calculation or running keywords. In the event that no REGEX is found, our implementation will assume that the tokens' pattern is not able to be calculated. At this point, safety checks have been put in place such that this request could not be processed.

**Overall Quality:**

The program's overall quality is properly functioning. Below is provided the subsets of functionality in the RED-YELLOW-GREEN template.

(GREEN) RPN-RUN

This corresponds to our main file, rpn.rb, which runs the program. Program calls all required source code and runs properly.

(GREEN) CHECK-ARGUMENTS

Arguments to decide whether to run in file mode or REPL mode work as expected.

(GREEN) FILE-MODE

When file(s) is/are passed as argument(s), the program properly runs in file mode as expected.

(GREEN) REPL-MODE

When no argument is passed, the program properly runs in REPL mode as expected.

(GREEN) STR-TO-TYPE

The input string from file or user input is converted into appropriate types.

(GREEN) KEYWORDS

Keywords are recognized, unknown keywords are considered invalid.

(GREEN) OPERANDS-OPERATORS

Operands, operators, and variables are stored appropriately

(GREEN) CALCULATIONS

All files/methods that relate to reverse polish notation calculations work as expected

**Areas of Concern:**

Upon completion of the implementation and testing portion of the deliverable, there are no known defects in our program.

Initially, we focused on handling all base cases first, then the edge cases, and lastly the corner cases as described in the project. Approaching the project this way helped handle the implementation portion because the debugging stage was minimized. According to our implementation, the user will be notified for all unexpected cases that will result.

With all things considered, there is a single unresolved Rubocop error in file_mode.rb
The error is provided below:
"C: Metrics/AbcSize: Assignment Branch Condition size for file_loop is too high. [17.5/15]"

In addition, there was another Rubocop error that has been detected. This error is quite unique in the sense that it is only found on Windows systems. The error is provided below:
"Layout/EndOfLine Carriage return character missing."

An important note on this is that, when running on MacOS, this error does not appear.

**Testing Strategy (98.68% Code Coverage):**

The tests we chose to write were at the unit level, so we decided to write methods that would easily be able to be tested with unit testing. Some methods had many options of outputs, thus needed multiple tests validating each case was necessary. Edge cases were considered, but due to the implementation of regex, most were thoroughly handled (Testing was done to ensure this first of course). This is how we achieved 98.68% code coverage (measured in simplecov). After completing the deliverable, it is safe to say that about 60% of time and effort spent went into deciding what unit tests to write and testing the assertions. The rest went into refactoring methods to be easily compatible with unit tests. We discovered that these methods had to be refactored from exploratory testing, because we experienced difficulties in implementing unit test to methods that called outside methods, without having to do a stub/mock.