ECS 140A – Programming Languages

Due: October 15, Friday, 2021, 11:59pm PT

This project specification is subject to change at any time for clarification.

Read carefully:

- The project has two parts. The second part is an extension of the first. You only need to submit one copy of your code.
- Download the rpn.h and rpn.cpp files for the started code.
- You **must** submit the source files and a README.txt file in a zipped folder.
- You should not use existing source code as a primer that is currently available on the Internet. You **must** specify in your documentation any sources of code that you have viewed to help you complete this project. Excessive collaboration, or failure to list external code sources will result in the matter being referred to Student Judicial Affairs.
- You **must** specify in your README file on how to compile and run your code.

Part 1

Write a C++ program called `rpn` to implement a reverse polish notation calculator.

- For more information on the notation see: `https://en.wikipedia.org/wiki/Reverse_Polish_notation`.
- Your implementation may use C++ 2011 standard, and any of the STL containers.
- `rpn` takes two inputs. The first input is an array of strings representing the formula in reverse polish notation. Each string is either an operator or an operand. The second one is the length of the array.
- The operands may be any floating number, "2", "-2", "3.5", etc.
- The returned result is a double precision floating number.
- The following operations must be implemented: add ($+$), subtract ($-$), multiply ($*$), divide ($/$), exponentiation ($**$), floor ($<$) and ceiling ($>$).
- The program must catch any exceptions and display the appropriate error message.
- Use standard input and output for this program.

Example:

- Input:

  - $[\text{"2"}, \text{"12"}, \text{"6"}, \text{"}-\text{"}, \text{"/"}, \text{"5"}, \text{"3"}, \text{"}+\text{"}, \text{"}*\text{"}]$ (an array of strings)
  - 9 (an integer)

- Output: 2.6667 (or $\frac{8}{3}$)

Part 2

Modify the rpn program from the first part to allow for an optional output. The "-l" option will display a listing of the input values in the following format: (operator operand1 operand2). Each operand should be indented one level greater than its operator, and trailing parens should be at the equivalent level of indentation to its match.

Example: The output of the example input in Part 1 with the -l opetion should be:

```
(*
  (/
    2
    (-
      12
      6
    )
  )
  (+
    5
    3
  )
)
```