Assignment – Reverse Polish Notation

Learning Goals

1. Develop your understanding of a "Stack" data structure.
2. Develop your understanding of parsing to break input into tokens.

The Task

Implement a Reverse Polish Notation calculator (also called a Postfix Notation Calculator).

The basic algorithm follows.

- While there are input tokens left
    - Read the next token from input.
    - If the token is a value
        - Push it onto the stack.
    - Otherwise, the token is an operator
        - If there are fewer than 2 values on the stack
            - **(Error)** Insufficient parameters.
        - Else
            - Pop the top 2 values from the stack.
        - Evaluate the operator, with the values as arguments.
        - Push the returned results back onto the stack.
- Once there is no more input,
    - If there is only one value in the stack that value is the result of the calculation.
    - Otherwise, if the stack has more than 1 value, **(Error)** The user input has too many values on the stack.

What is a token?

A token is a logical piece of text that makes up the input stream. For example, given the input line “49 120 +”, the tokens are 49, 120, and +. Note that we logically group the 4 and 9 together to create a single 49 token.

The Challenge

You are reading characters one at a time from input. You will need to convert the sequence of characters “125” to the integer 125. We did this once before together in class. The process requires the help of some ASCII character constants.

Implementation

You will create a file **rpn.c**. You need only one function, main, though you may choose to create some helper functions as well. The function **main** will read characters from the input one at a time and implement the algorithm above.

Include **stack.h** into **rpn.c**. Use the stack functionality provided by stack.h when implementing **rpn.c**, including: push, pop, isStackEmpty, getStackSize, and emptyStack. Do not recode any of this stack functionality.

Test Cases

Compilation:

```
gcc -o rpn rpn.c stack.c
```

Valid test cases:

```
$ ./rpn < valid1.in
Result: 30.00

$ ./rpn < valid2.in
Result: -5.50

$ ./rpn < valid3.in
Result: 31.75
```

Invalid test cases:

```
$ ./rpn < invalid1.in
There are too few operators in the expression.

$ ./rpn < invalid2.in
Input error at character 14 (-); too few arguments on
stack.

$ ./rpn < invalid3.in
Stack full when pushing (6) at position 241
```