Havish Chennamraj
hchennamraj3@gatech.edu

CN Programming Assignment 1 – RPN Calculator

Name and description of files submitted

1.) rpnserver.py - Contains Server code; Server listens on a particular port and accepts requests from clients on the said port. The request is a Postfix expression strictly following the format "OP1 OP2 Operator". The job of the server is to compute this expression (accommodates computation of decimal values as well) and return the result of the computation back to the client.

2.) rpnclient.py - Contains Client code; Client connects to the port on which the server if listening and accepting requests. The Job of the client is to request the Server to compute a particular Postfix Expression. It also validates the correctness of the Postfix expression. For instance checking if the given expression is in a valid Reverse Polish Notation. It also checks if the operands being supplied in the expression are valid numbers (float/int) or not, and if the operators are strictly one among "+, -, * and /"

Instructions for running the client and the server

1.) chmod +x rpnserver.py
2.) chmod +x rpnclient.py
3.) chmod +x killserver.sh
Above steps have to be run only once while setting up to set execute permissions to the python scripts

To Launch the server

“./rpnserver <PORT_NUMBER>”
For instance, “*./rpnserver 12345”*

To Launch the Client

./rpnclient <PORT_NUMBER> "<VALID_POSTFIX_EXPRESSION"
For instance, './rpnclient 12345 "25 5 /"'

The Client automatically exits after its request is processed.
To kill the server, send a CTRL+Z signal to stop the server process and then run "./killserver.sh"

Protocol Description

1.) Format of messages exchanged between Client and Server

Client to Server: It is always assured that the Client sends a "valid" Postfix expression to the Server in the following format:
"Operand1 Operand2 Operator", where Operand1 & Operand2 are valid Intergers or Decimals and the Operator is one among '+', '-', '' & '/'.*

Server to Client: The Server sends the result of the computation to Client. If the evaluation of the expression is successful, the Server sends the result back as a String. If the execution on the Server results in some error, the Server simply sends the error back to the Client.
Successful Evaluation: "<result>"
Unsuccessful Evaluation: "<error>"

2.) How are multiple compute requests handled between Client and Server?

- Server is unware of the state of expression evaluation. The job of the server is to simply process the request it receives and return back the result as a string.
- State of the expression evaluation is maintained by the client. Based on the stage of the given Postfix expression evaluation, the client decides which two operands and operator to send to the server. The Client achieves this by doing the following:
    1.) After validating the Postfix Expression, the client splits the expression using space as a delimiter. It is important to note that the Client recognizes only Space as a delimiter in the expression. In other words,

the user cannot give an expression separated by commas and expect a result back. The client would throw an error in such cases.

2.) The client maintains a global pointer to access the elements in the array obtained after splitting the expression with white space. For simplicity sake, lets call this array "expSplit". The client also maintains a global stack "operandStack" to maintain the order of execution of the PostFix expression
3.) The client moves the global pointer (initialized to 0) from left to right; If the element encountered is an operand, the client pushes it into the operand stack. If the element encountered is an operator, the client pops the top 2 operands from the stack, parses the expression as "Operand1 Operand2 Operator" and sends the request to the server.
4.) The Client pushes the result back from the server into the Operand Stack and continues in the same fashion as mentioned in step 3.

3.) How TCP Server knows the expression is complete?

- Like mentioned before, the Server is completely ignorant of the state of Postfix expression evaluation. The only job of the server is the process the request it has received from the client and return the result back.

- The Client maintains the state of Evaluation. Like mentioned before, the client has a global pointer to process the expression sequentially. Also, a valid Postfix expression is bound to end with a operator. The Client uses this and its global pointer to figure out if the request its about to send is the last request of the expression or not.

4.) Other Implementations

1.) Validation of Port Numbers: Both the server and the client do basic validation on the <PORT_NUMBER> provided as a command line argument: Check whether <PORT_NUMBER> is a valid number and lies between 1024 & 65535
2.) Apart from validating <PORT_NUMBER>, the client also validates the given Postfix expression. The client checks if all the operands are valid integers or decimal point numbers, and the Operator is one among '+', '-', '*' & '/'. It also checks if the given expression is in a valid Reverse Polish notation or not.

3.) The server is very persistent. Any error it encounters during its execution, the server just returns the error back as the result of the execution. The client decodes the response from the server and decides on its next move. For instance, in case of division by zero, the server responds with an appropriate error message. The client on decoding the error terminates the execution.
4.) There server is perfectly capable of handling operations on decimal values with good precision.
5.) There is no limit to the number of input tokens one can give in the expression, the overflow issues are handled inherently by python.

Limitations

1.) No bugs to the best of my knowledge
2.) The communication between client and server and vice versa can handle only a maximum of 1024 bytes.
3.) Expressions giving abnormally high values would result in server sending back 'inf'. For instance, 10^1000 - 10 would result in client receiving 'inf' from server.
4.) Client supports only white space as a delimiter for Postfix expression. Any other delimiter would result in client throwing an error.
5.) If the postfix expression is invalid, the client doesn't dispatch the request for the valid portion of the postfix expression. For instance, in "120 40 + 30a / /" although "120 40 +" is valid, it is not processed and the client is terminated. \