CS 6250 - Computer Networks I
Fall 2018
Date : 30th October 2018
Title : Programming Assignment

Name : Archit Bansal
Email : abansal88@gatech.edu
GTID : 903396126

Files and descriptions:

1. rpnserver.py : This file contains the python implementation of the server mentioned in the problem statement. The server expects a postfix expression with a single operator, operates it and sends back the result to the client. The server is capable of performing operations on floats and ints as well.
2. rpnclient.py : This file contains the python implementation of the client mentioned in the problem statement. The client accepts a user input, validates it against various rules and sends to the server in chunks for processing. The client saves and displays the intermediate results that the server responds with in every iteration.
3. README.pdf : This file contains the details of all the files in the project.
4. sample.txt : This file contains various inputs and the corresponding outputs at server and client demonstrating end to end runs of the server.

Compiling and running the client and server:

1. Server : Download the file rpnserver.py and give it executable permissions using the following command:
   chmod +x rpnserver.py
2. Once the executable permissions are given to the file it can be run as shown:
   ./rpnserver.py <port_number>
   where port_number is the port of the server where it listens on.
   If the port_number is not provided, it starts on the port 65432 by default.

   To kill the server : Ctrl + c

3. Client : Download the file rpnclient.py ang give it executable permissions using the following command:
   chmod +x rpnclient.py
4. Once the permissions are given, run it as follows:
   ./rpnclient.py <port_number> <postfix_expression>
   Where port_number is the port on which to connect to the server, and **postfix_expression is the postfix expression to be evaluated separated by spaces in between the tokens.**
   For eg : “23 4 * 5 +” is a valid input which translates to 23 * 4 + 5. **If there is no space in between the tokens, the input will be marked as invalid.**

   The client exits itself when the whole expression is evaluated.

Protocol description:

Functionalities:

Server : The server is listening for connections on the specified port and expects an input with two operands and an operator to be fed to it from the client. The server then operates the required operation on the input and returns with the return value in the happy case. In case of any error, it notifies the client about it through the response to the request.

Client : Client takes in the postfix expression and the port to connect to on the server. It performs various validations on the input (mentioned in the next section). It then sends operations one at a time to the server, saves the result and repeat till the full postfix expression is evaluated. If the server returns an error, the client passes the error to the user and terminates the connection with the server.

Communication:

The server and client communicate among each other with specified message formats as described below:

A. Client to Server :
<Operand1> <Operand2> <Operator>
Where Operand1 and Operand2 are valid float point values as per python

specifications, and Operator belongs to {+, -, *, /}

B. Server to Client :
<Return Value>, in happy case
<"error : <error_description>">, in case there were errors while processing
If the client sees the keyword "error" in the server response, it catches the error and closes the connection displaying the relevant error to the user.

Multiple compute requests:
For evaluating a single postfix expression, there are multiple interactions between the client and the server. The client sends a single request to the server for every instance where it has two operands and an operator to compute, saves the result sent by the server as an intermediate value and uses the same for sending further requests to the server. The same process continues till the client is done with full postfix expression after which it closes the connection.

Termination:

Once the full postfix expression is evaluated, the client closes the socket from its end. The server who is waiting to read data on recv() call gets a None value, breaks out of the loop, closes the connection on its end and continues listening for more connections.

Bugs and limitations:

1. There are no known bugs in the program. However, there are some validations implemented in the client that the user should know about. The validations are as follows:

   A. The maximum allowed length for the postfix expression is 1024.
   B. The input should be a valid postfix expression.
   C. None of the operands must be illegal float values ,i.e., 11.3a and 12.e etc will generate an error.
   D. If at any point during calculations, the answer at the server exceeds 1024 in length, the server sends an error message ("overflow in the calculation") instead of the retVal.
   E. If any point in the calculation a divide by 0 is encountered, it is displayed to the user and further computations are stopped.