CSE250 Project #2

Sheng Liu

June 2021

1 Introduction

You use stacks and queues to solve a real problem, *i.e.*, expression evaluation, in this project. Given a valid expression, *i.e.*, a string, you need to write code that computes the output value of the given expression. For example, given "`min(2, 6) * 5 / 2 + 18 / max(3, 1)`", your code should return an integer 11. Given an invalid expression, your code should return -1024. For example, given "`min(2, 6 * 3`" (this expression is invalid because there is no right parenthesis ")"), your code should return -1024.

I will talk about how you can evaluate expressions using stacks and queues next Monday. Be sure to watch the video or attend the live lecture, as there are no single resources that tells you how to do this and some online resources you can find contain errors!

2 Details

You can make the following assumptions (in other words, the test cases will follow the following rules):

- All operands in the expression are integers. Strings like "`12.2, 12.`" will never appear in the expression.
- There are only two functions that might appear in the expression, "`min, max`". They refer to the two functions defined in std namespace.
- "`{}, []`" will never appear in the expression. If there are parenthesises in the expression, they will always be "`(, )`".
- Only white space, comma, "`min, max, (, ), 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, +, -, *, /`" may appear in an expression.
- The only reason an expression can be invalid is that its parenthesises does not match, *e.g.*, there are more left parenthesises than right parenthesis.

If you do not know what the output value should be, you can compute its value using C++ code. For example, given "`min(2, 6) * 5 / 2 + 18 / max(3, 1)`", you can define write the follow function to compute its value if you do not know how to compute its value manually. `int x = min(2, 6) * 5 / 2 + 18 / max(3, 1);` The value of `x` is the desired output.

3 Functions

A good reference:
https://runestone.academy/runestone/books/published/pythonds/BasicDS/InfixPrefixandPostfixExpressions.html
It explains how an expression can be evaluated in detail and also explains some terms, *e.g.*, infix, postfix.

- `bool isValidExpression(const string &exp)`: checks whether the input expression is valid or not. If it is invalid, return false. Otherwise return true.
- `queue<string> parseExpression(const string &exp)`: splits an expression into strings that are function names, operators, operands, parenthesises and commas. For example, given `min(2, 6) * 5 / 2`, this function returns a queue containing `min | ( | 2 | , | 6 | ) | * | 5 | / | 2` (white spaces and — are used to separate elements in the queue). The order of the elements matter.
- `queue<string> inFixToPostFix(queue<string> &exp)` converts an infix expression to a postfix expression.
- `int EvalPostFixExp(queue<string> &exp)`: takes as input a postfix expression and returns the output value of the postfix expression.

4 Submission

You **only** need to submit the cpp file named "evalExp.cpp" to AutoLab. Do **NOT** modify the code we provided in the cpp file or use (include) any additional function defined in C++ std namespace. As your projects will be graded automatically, modifying the provided code may cause our grading script to fail. If this happens, the grade will be **0**. Grades for the projects that use additional std functions will also be **0**. Our grading script will call the functions you implement in the cpp file and check whether return value is correct. As a few students attempted to guess our test cases by making tens of submissions for project 1, the **maximum** number of submissions you can make for this project is **10**.

5 Wikipedia

https://en.wikipedia.org/wiki/Shunting-yard_algorithm is an OK reference. It clearly explains the basic ideas. However, its way of dealing with functions is not 100% correct. If you strictly follows it. Your code may fail if the input expression contains "`min, max`". However, your code should be fine for expressions that does not contain functions.