Assignment 3 : Arithmetic Expression Evaluation using Stack

Deadline: 24/11/2020 11:55 PM

Instruction:

1. Write your code in the c file named "expr_eval.cpp".
2. Avoid plagiarism. If you are found to adopt any unfair means you will get a straight 0.
3. Upload your code (only the cpp file) in elms.
4. Deadline is 15/12/2020 11:55 PM.

Task:

In this assignment you will get an idea how compilers evaluate arithmetic expressions. You have to evaluate an arithmetic expression represented by a string. The expression can contain parentheses **'('** or **')'**. You can assume parentheses are well-matched. For simplicity, you can assume all the numbers are **single digit** numbers and only binary operators allowed are **+**, **-**, *****, and **/**. For example, 1+2*6-2+(9-6).

For this assignment, an arithmetic expression will contain symbols of three types: digits, operators and parentheses. To evaluate the expression you will need two stacks, one for the numbers: we will call this the **value stack**, and another for the operators and parentheses: we will call this the **operator stack**. The arithmetic expression is to be taken as a string input (with no spaces in between). We will process the arithmetic expression character by character. The algorithm required to process the arithmetic expression is delineated below.

1. While there are still characters left in the string,
    1.1 Get the next character.
    1.2 If the character is:
        1.2.1 A digit **'0'** to **'9'**: push it onto the **value stack**.
        1.2.2 A left parenthesis **'('**: push it onto the **operator stack**.
        1.2.3 A right parenthesis **')'**:
            1.2.3.1 While the thing on top of the **operator stack** is not a left parenthesis **'('**,
                1.2.3.1.1 Pop the operator from the **operator stack**.
                1.2.3.1.2 Pop the **value stack** twice, getting two operands.
                1.2.3.1.3 Apply the operator to the operands, in the correct order.
                1.2.3.1.4 Push the result onto the **value stack**.
            1.2.3.2 Pop the left parenthesis **'('** from the **operator stack**, and discard it.
        1.2.4 A **'+'** operator or a **'-'** operator:
            1.2.4.1 While the **operator stack** is not empty, and the top thing on the operator stack is not a left parenthesis **'('**,
                1.2.4.1.1 Pop the operator from the **operator stack**.

1.2.4.1.2 Pop the **value stack** twice, getting two operands.
1.2.4.1.3 Apply the operator to the operands, in the correct order.
1.2.4.1.4 Push the result onto the **value stack**.
1.2.4.2 Push the **'+'** or **'-'** onto the **operator stack**.
1.2.5 A **'*'** operator or a **'/'** operator:
1.2.5.1 While the **operator stack** is not empty, and (the top thing on the operator stack is a **'*'** or a **'/'**),
1.2.5.1.1 Pop the operator from the **operator stack**.
1.2.5.1.2 Pop the **value stack** twice, getting two operands.
1.2.5.1.3 Apply the operator to the operands, in the correct order.
1.2.5.1.4 Push the result onto the **value stack**.
1.2.5.2 Push the **'*'** or **'/'** onto the **operator stack**.
2. While the **operator stack** is not empty,
2.1 Pop the operator from the **operator stack**.
2.2 Pop the **value stack** twice, getting two operands.
2.3 Apply the operator to the operands, in the correct order.
2.4 Push the result onto the **value stack**.
3. At this point the **operator stack** should be empty, and the **value stack** should have only one value in it, which is the final result. Pop it and print the value.

- The stack data structure is already implemented and ready to be used. It is designed in a way to store int type data. Although **value stack** and **operator stack** are to be used for numbers(int) and operators(char) respectively, you don't need to worry about it. You can store a char in an int. Just remember to subtract '0' from a digit(char) to get it's integer value ('3'-'0' = 3, '5'-'0' = 5, etc) when you push a digit(char) from the expression into the **value stack** (see 1.2.1). For the **operator stack** you can directly push in and pop out a char.
- Use the int peek() method of the stack data structure to see the top item without removing it from the stack.
- A function int applyOp(int lvalue, int rvalue, char op) is implemented for facilitating the application of the operator to the operands. Here 'lvalue' is the left operand, 'rvalue' is the right operand and 'op' is the operator.
- Here "in the correct order" phrase means that the first value popped from the **value stack** will be the right operand and second value popped will be the left operand.
- If you haven't already noticed, in 1.2.5.1 the conditional statement of the while loop contains a bracket around the statement "(the top thing on the operator stack is a **'*'** or a **'/'**)" which means that the logical OR operation must be done before the logical AND operation.
- The assignment may seem tough at first sight, but the experience of completing it by yourself will be a greater achievement.