Parenthesis Matching Problem Using Stack Data Structure (Applications of Stack)

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-32

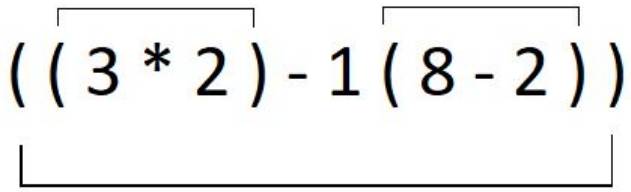
I was very excited to bring this topic to your attention. Parenthesis matching is one of the basic applications of the stack we learned about in our last ten lectures. This will be thought-provoking to you all as well. Since the dawn of programming, parenthesis matching has been a favorite topic. It is a must learn. So, today, we'll start learning about parenthesis matching and how it gets implemented using stacks.

Parenthesis matching has always been threatening to beginners. But realizing its implementation using stacks makes it very intuitive and easy to deal with.

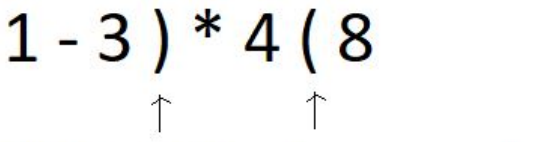
What is parenthesis matching?

If you remember learning mathematics in school, we had BODMAS there, which required you to solve the expressions, first enclosed by brackets, and then the independent ones. That's the bracket we're referring to. We have to see if the given expression has balanced brackets which means every opening bracket must have a corresponding closing bracket and vice versa.

Below given illustrations would surely make it clear for you.



Balanced Parentheses



No coresponding opening bracket

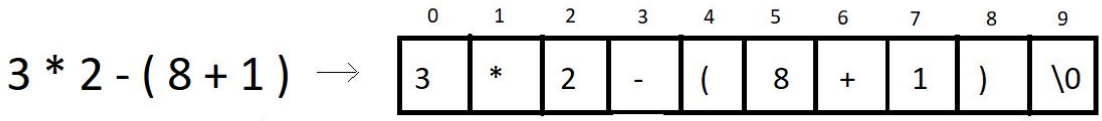
No corresponding closing bracket

Unbalanced Parentheses

Checking if the parentheses are balanced or not must be a cakewalk for humans, since we have been dealing with this for the whole time. But even we would fail if the expression becomes too large with a great number of parentheses. This is where automating the process helps. And for automation, we need a proper working algorithm. We will see how we accomplish that together.

We'll use stacks to match these parentheses. Let's see how:

1. Assume the expression given to you as a character array.



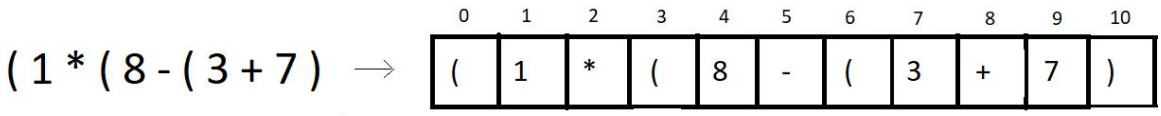
2. Iterate through the character array and ignore everything you find other than the opening and the closing parenthesis. Every time you find an opening parenthesis, push it inside a character stack. And every time you find a closing parenthesis, pop from the stack, in which you pushed the opening bracket.

3. Conditions for unbalanced parentheses:

- When you find a closing parenthesis and try achieving the pop operation in the stack, the stack must not become underflow. To match the existing closing parenthesis, at least one opening bracket should be available to pop. If there is no opening bracket inside the stack to pop, we say the expression has unbalanced parentheses.
- For example: the expression **(2+3)*6)1+5** has no opening bracket corresponding to the last closing bracket. Hence unbalanced.
- At EOE, that is, when you reach the end of the expression, and there is still one or more opening brackets left in the stack, and it is not empty, we call these parentheses unbalanced.
- For example: the expression **(2+3)*6(1+5** has 1 opening bracket left in the stack even after reaching the EOE. Hence unbalanced.

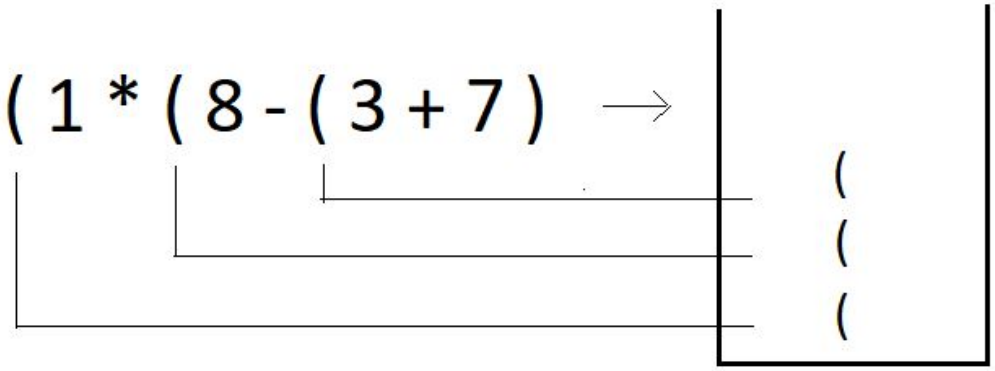
4. Note: Counting and matching the opening and closing brackets numbers is not enough to conclude if the parentheses are balanced. For eg: **1+3)*6(6+2**.

Example:

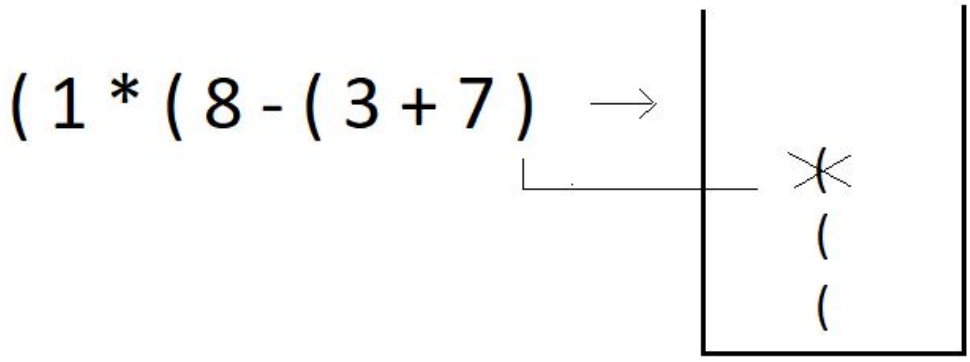


We'll try checking if the above expression has balanced parentheses or not.

Step 1: Iterate through the char array, and push the opening brackets at positions 0, 3, 6 inside the stack.



Step 2: Try popping an opening bracket from the stack when you encounter a closing bracket in the expression.



Step 3: Since we reached the EOE and there are still two parentheses left in the stack, we declare this expression of parentheses **unbalanced**.

I have one task for you as well. Try checking if these expressions are balanced or not. And also, tell the number of times you had to push or pop in the stack. Also, comment on the time complexity of this algorithm. Answer the best and the worst runtime complexity for an expression of size n.

1. **7 - (8 (3 * 4) + 11 + 12)) - 8)**

I would just recommend everyone to try coding the algorithm for parentheses matching on their own, and do tell me if you could. We'll see this segment in the next tutorial anyways. Do let me know if you are enjoying the course. Bigger things are waiting, just stay!

I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial, where we'll learn to automate this process of parentheses matching using stacks in C. Till then, keep coding.