Question 1

Solution

To solve this problem, my thought is simple. For each character in the string, put it into the stack if it is an open bracket, and take the last element out of the stack if it is a close bracket and it is the same type as the last element in the stack. Return false in cases:

1. The current character is a close bracket and the stack is empty .

2. The current character is a close bracket but the last element in the stack is not the corresponding open bracket.

3. After accessing every characters, the length of the stack is not 0.

For implementing the steps mentioned above, the class of stack contains 2 varibles and 3 functions:

Varible length: int type, represents the length of the stack

Varible content: list type, represents the elements of the stack

Function in_: (requires an input element) add the element into the stack (it's tough for a non-native speaker to come up with a good name to describe this function, and since the name "in" is used by python, I named it "in_")

Function out: (no input required) take away the last element in the stack and return True. Return False if the stack is empty

Function last_element: (no input required) return the last element in the stack. Return False when the stack is empty.

Complexity

Time Complexity: O(n)

For almost every cases, the core loop runs n times (n is the length of the string). The complexity of the code inside the loop is O(1), since they either return bool varibles or simply assign values. Thus the overall time complexity is n*O(1) which is O(n)

Space Complexity: O(n)

The stack stores at most n/2 elements in the worst case, and since extra storage grows linearly with input size, the space complexity is O(n).

Question 2

Solution

To address this problem, I coded a loop to extract numbers and “+”, ”-”, ”*”, ”/” from the string and store them into a list. I didn’t do this at the first trial and I soon noticed that numbers like “15” will be put into the stack as two elements “1” and “5”.

Then, I implemented another loop to iterate through each element in the list. If the current element is a number or a number with a “-“ in front, it will be added into the stack as an integer. However, if the current element is a and “+”, ”-”, ”*”or ”/”, the program will run the corresponding calculation for the last element and the element before the last one.

Finally, these two elements will be poped from the stack and the outcome of the compute will be pushed into the stack. After the end of this loop, the program ultimately return the first element in the stack since the test cases don’t invole invalid expressions, and for valid postfix expressions, the rest elements will be poped gradually, and the first element will be the result.

Complexity

Time Complexity: O(n)

There are two loops in my solution, which both are O(n) since they iterate through elements in the string, and their sum is O(n) as well.

Space Complexity: O(n)

In the worst case, the stack stores n/3 varibles, where n is the length of the string. Since extra storage grows linearly with input size, the space complexity is O(n).