1. **Expressions converted from infix to postfix:**
    - a b + c d + * a + c * 6 - b *
    - y x 3 - z * + +=
    - max a b > a b ?: =

2. **Explain how to implement two stacks in a single array with n elements in such a way that neither stack overflows unless the total number of elements in both stacks together is n. The push(item) and pop() operations should still run in time O(1). Explain why they do.**
    - You create two stacks that share the same array. Ideally both stacks should be defined within a Class that ties them both together.
    - Each stack should have a property tracking its size, *k*
    - Each Stack stacks elements from opposite ends of the array. The number of elements contained in each stack is represented as *k. (*Adding both *k's* yields *n)*
        - **e.g**, Stack A will push from left to right, so that the location of the top of the Stack is essentially Arr[$k - 1$].
        - **E.g,** Stack B will push from right to left, so that the location of the top of the Stack is Arr[$n - 1 - k$]
    - If there is empty space in the array between both tops, then that means there's room to push to the stack and array. But if the difference is zero, that means there will be an overflow, since doing so would exceed *n*
    - The reason why `push` and `pop` operations are constant time, is that we have direct access to the array, and we can deduce the locations of where to pop and where to push based on each stack's size with simple arithmetic.
3. **Show how to implement a queue using two stacks. Analyze the running time of the queue operations enqueue(item) and dequeue(). Explain your analysis.**
    - There will be two stacks, Stack A, Stack B, and a Queue.
    - New element is added to top of Stack A when doing enqueue operation
    - When doing de-queue operation, if Stack B is empty all elements are moved from Stack A to Stack B, and the top of Stack B is then popped.