CSE 4304-Data Structures Lab. Winter 2021
Lab-02
Date: May 21, 2021 (Friday)
Target Group: All Lab groups
Topic: Stacks

Instructions:
- Task naming format: fullID_T01L02_2A.c/cpp
- If you find any issues in problem description/test cases, comment in the google classroom.
- If you find any test case that is tricky which I didn't include but others might forget to handle, please comment! I'll be happy to add.
- Modified sections will be marked with BLUE colour.

Task-01(a)

Stacks are dynamic data structures that follow the ***Last In First Out*** (LIFO) principle. The last item to be inserted into a stack is the first one to be deleted from it. For example, you have a stack of trays on a table. The tray at the top of the stack is the first item to be moved if you require a tray from that stack.
The Insertion and Deletion of an element from the stack are a little bit different from the traditional insertion/deletion. We define the two corresponding operations as ***Push()*** and ***Pop()*** from the stack.

The first line of input represents **N** which will be the size of the stack. The next input will be **T** representing the number of operations. Each time of the **T** operations, the user will choose as to whether s/he wants to do push/pop. If the operation is Push, the system will ask for one more ***item***. For each test cases, the system will show the state & size of the stack. If the stack is empty, the state will be **(size=0,items=NULL).**
For ease of implementation, let's assume that Push is represented by 1 and pop is represented by 2.

Sample Input	**Sample Output**

10
8
1 10
1 20
2
1 50
2
2
2
1 25
	size=1 items=10
	size=2 items=10 20
	size=1 items=10
	size=2 items=10 50
	size=1 items=10
	size=0 items=NULL
	size=0 items=NULL
	size=1 items=25

[**Note**: Don’t use any STL functions for this task. Your solution must include separate functions for push, pop, isEmpty, size operations.]

Task-01(b)

C++ offers a header file called <stack> which has built-in functionalities already implemented as library functions. Follow this link and try to understand the usage of each function. Finally, implement **Task-1** using those STL library functions.

Task-02

Write a program that will take **N** strings as input and print the reverse of them. There are many ways to solve this problem. However, today we want you to solve this problem using Stack.
Input will be the number of test cases (**N**) followed by N strings.

Sample Input	**Sample Output**
3 IUT data mozzarella	TUI atad allerazzom

Task-03

Write a program that will take an expression represented using postfix notation as input and evaluate the expression using stack.
Input expression may contain addition (+), subtraction (-), multiplication (*) and division (/) operators and numbers between (0~9).

Sample Input	**Sample Output**
2 345*+6- 225+*1+5-52**	17 100

Task-04

Write a program that will take a mathematical expression as input and check whether it is properly parenthesized or not.

The first line of input will take an integer **N** signifying the number of test cases. The next lines will be **N** mathematical expressions.Each input expression may contain any single-digit number (0~9), operators (+ - x /) and any parenthesis ( )/[ ]/{ }.
The output will be Yes/No representing whether it is properly parenthesized or not.

Sample Input	**Sample Output**
5	Yes
[5 + (2 x 5) - (7 / 2)]	No
[1 + { 3 x (2 / 3)] }	Yes
[(1 + 1)]	No
[(1 + 1])	Yes
[( )] { } { [( ) ( )] ( ) }	

Task-05

For sure, the love mobiles will roll again on this summer's street parade. Each year, the organisers decide on a fixed order for the decorated trucks. Experience taught them to keep free a side street to be able to bring the trucks into order.
The side street is so narrow that two cars can’t pass each other. Thus, the love mobile that enters the side street last must necessarily leave the side street first. Because the trucks and the ravers move up close, a truck cannot drive back and re-enter the side street or the approach street.
You are given the order in which the love mobiles arrive. Write a program that decides if the love mobiles can be brought into the order that the organisers want them to be.

Input

There are several test cases. The first line of each test case contains a single number **n**, the number of love mobiles. The second line contains the numbers ***1 to n*** in an arbitrary order. All the numbers are separated by single spaces. These numbers indicate the order in which the trucks arrive in the approach street. No more than 1000 love mobiles participate in the street parade.

Output

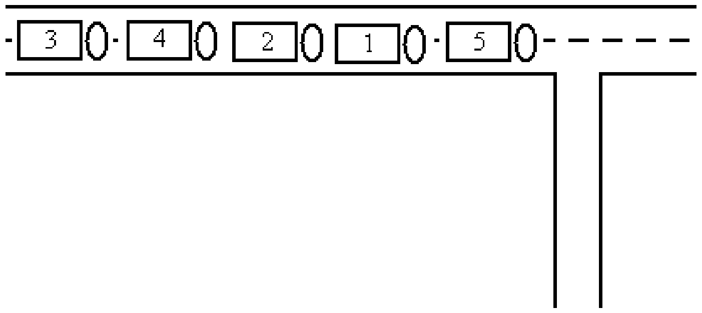
For each test case your program has to output a line containing a single word "yes" if the love mobiles can be re-ordered with the help of the side street, and a single word "no" in the opposite case.

Sample Input	**Sample Output**
5 5 1 2 4 3	Yes
5 5 4 3 2 1	Yes
5 1 2 4 3 5	No
5 1 2 5 3 4	Yes

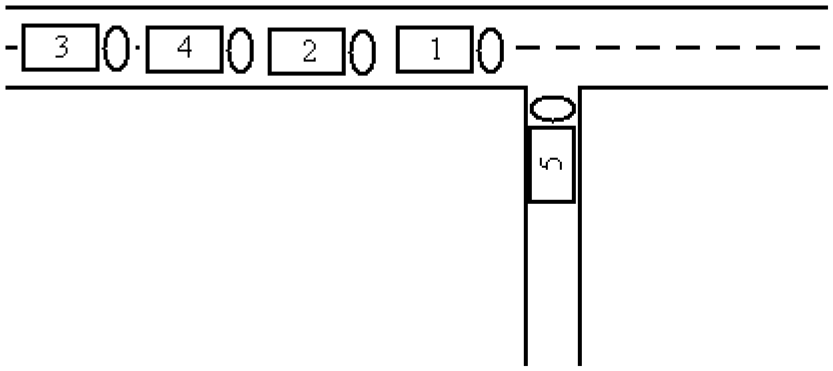
Note: You can submit your solution here. (Optional)
(Check next page for illustration)

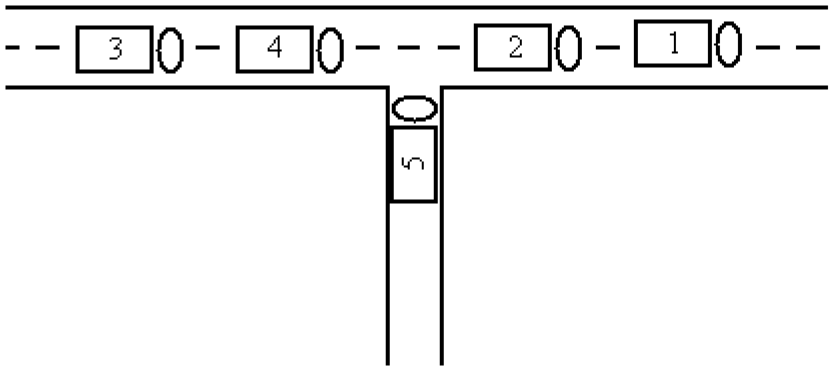
Illustration (for first test case):

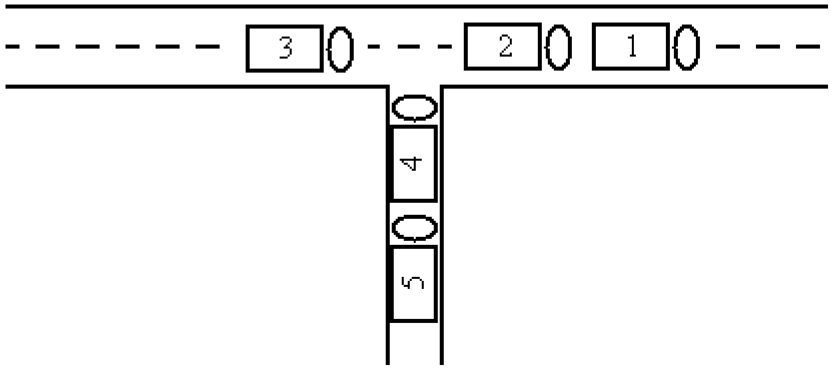
The sample input reflects the following situation:

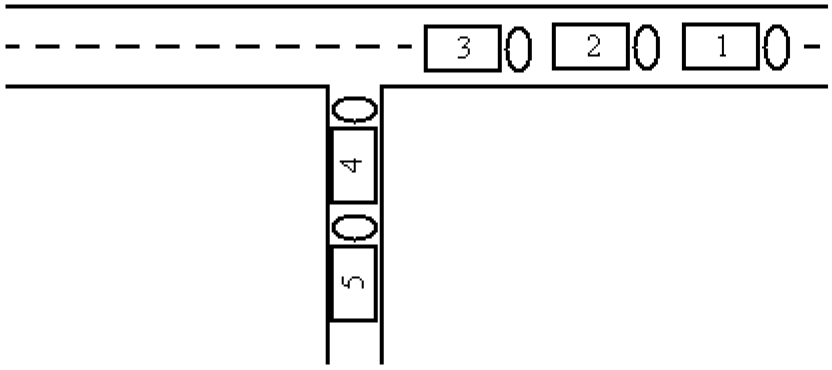


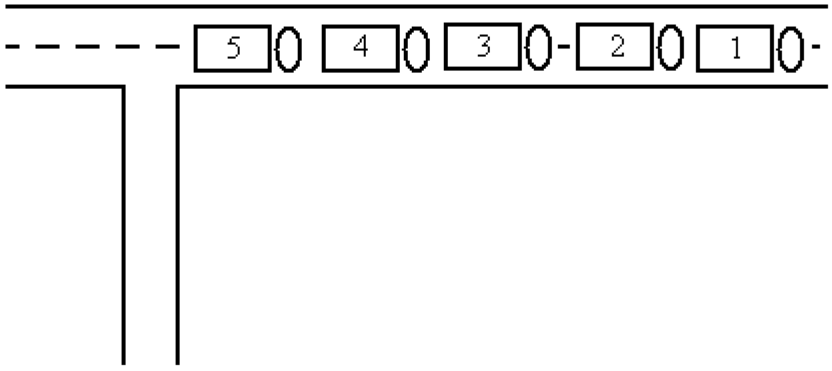
The five trucks can be re-ordered in the following way:






3
2
1
4
5


3
2
1
4
5


5
4
3
2
1

Task-06 (Optional, will carry bonus marks)

Give a string ‘**s**’ containing ‘**(**‘ and ‘**)**‘ parentheses, we need the minimum number of parentheses needed to be introduced at any position so that the resulting parentheses string is valid.

Sample Input	**Sample Output**
( ))) ((	4
(( )) (( ))	0
((( )) ( )	1
)))))) ((	8
(( ) ((	3

Task-07 (Optional, will carry bonus marks)

Transform the algebraic expressions with brackets into RPN form (Reverse Polish Notation) aka Post-fix notation. Two-argument operators: +, -, *, /, ^ (priority from the lowest to the highest)., brackets ( ). Operand: only letters: a, b, …. Z.

Input
t (the number of expressions <=100)
Then there will be t expressions (length <=400)

Output:
The expressions in RPN form, one per line.

Sample input	Sample Output
3 (a+(b*c)) ((a+b)*(z+x)) ((a+t)*((b+(a+c))^(c+d)))	abc*+ ab+zx+* at+bac++cd+^*

Note:

- You can submit your solution **here**. (Optional)
- A tutorial for converting infix notation to postfix can be found here.

<u>Task-08</u> (Optional, will carry bonus marks)
Given an array, print the ***Next Greater Element (NGE)*** for every element. The next greater element for an element ***x*** is the first greater element on the right side of ***x*** in the array. Element for which no NGE exist, consider next greater element as -1.

<u>Examples</u>:
For an array, the last element always has NGE as -1.
For an array sorted in descending order, every element has NGE as -1
For the input array (4, 5, 2, 25) the NGE for each element is (5, 25, 25, -1}. For the input array (13, 7, 6, 12) the NGE for each element is (-1, 12, 12, -1}

We can solve this using two simple nested loops! The outer loop picks the elements one by one and the inner loop looks for the first greater element for the element picked by the outer loop. If a greater element is found then that element is printed as the NGE, otherwise, -1 is printed.
But this approach has the worst-case Time complexity is $O(n^2)$. (worst case occurs when the elements are in descending order. It will lead us to look for all the elements.)

We can improve the time complexity to O(n) using stacks! Your task is to propose an algorithm to find the NGE of every element using stack in O(n) time.

<u>Input</u>:
First-line will take the value of **N** which represents the number of test cases. Afterwards, there will be N test cases. Each test case can consist of a different number of integers. Every test case will end with a -1 indicating the end of a particular test case (-1 will not be considered as part of the input.)

<u>Output</u>:
Find the NGE of every element using stack with O(n) worst-case time complexity.

Sample input	**Sample Output**
6	5 25 25 -1
4 5 2 25 -1	-1 12 12 -1
13 7 6 12 -1	13 21 -1 20 -1
11 13 21 3 20 -1	17 18 5 7 2 7 7 18 25 -1 -1 -1 -1
12 17 1 5 0 2 2 7 18 25 20 12 5 1	2 -1
2 -1	20 30 40 50 -1
10 20 30 40 50 -1	-1 -1 -1 -1 -1
50 40 30 20 10 -1	

Hint: Check the solution from here only if you fail to do it even after several attempts.

More practice problems:
https://www.codechef.com/problems/COMPILER
https://www.spoj.com/problems/JNEXT/