CSE 4304-Data Structures Lab. Winter 2022-23
Batch: CSE 21
Date: August 21, 2023
Target Group: All
Topic: Stacks

Instructions:

- Regardless you finish the tasks in the lab, you have to submit the solutions in the Google Classroom. In case I forget to upload the tasks there, CR should contact me. The deadline will always be at 11.59 PM of the day in which the lab has taken place.
- Task naming format: fullID_T01L02_2A.c/cpp
- If you find any issues in the problem description/test cases, comment in the google classroom.
- If you find any test case that is tricky that I didn't include but others might forget to handle, please comment! I'll be happy to add.
- Use appropriate comments in your code. This will help you to easily recall the solution in the future.
- Obtained marks will vary based on the efficiency of the solution.
- Do not use <bits/stdc++.h> library.
- Modified sections will be marked with BLUE color.
- **You are allowed to use STL stack for tasks 2, 3, and 7.**
- Two ways to take inputs if number of input elements is not specified:
    - Taking inputs until End-of-File (EOF) is reached
      (Press **Ctrl+Z** after typing in your inputs. You should see something like **^Z** for Windows.)

```
int n;
vector<int>a;
while(cin >> n)
{
    a.push_back(n);
}
```

    - Taking inputs until "Enter" button is pressed

```
char c;
int n;
vector<int>a;
while (true) {
    c = cin.peek(); // Peek at the next character

    if (c == '\n') {
        // Exit loop when Enter is pressed
        break;
    }
    cin >> n;
    cin.get();  // Consume the character from the stream
    a.push_back(n);
}
```

Group	**Tasks**
2A	1 2 3 7
2B	1 2 4 6
1B	1 6 8 10
1A	1 4 8 10
Assignments (all)	**1-13 (except the once you did in the respective labs)** **Plagiarism will be checked. Pls be careful.**

Task-01: Implementing the basic operations of Stack
Stacks is a linear data structure that follows the Last In First Out (LIFO) principle. The last item to be inserted is the first one to be deleted. 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 slightly differ from the traditional operation. We define the corresponding operations as Push() and Pop() from the stack.

The first line contains *N* representing the size of the stack. The lines contain the 'function IDs' and the required parameter (if applicable). Function ID 1, 2, 3, 4, 5, and 6 corresponds to push, pop, isEmpty, isFull (assume the max size of Stack=5), size, and top. The return type of inEmpty and isFull is Boolean. Stop taking input once given -1.

Input	Output
5 3 2 1 10 1 20 5 1 30 6 2 1 40 1 50 4 1 60 4 5 1 60 5 2 6 -1	 True Underflow 10 10 20 2 10 20 30 30 10 20 10 20 40 10 20 40 50 False 10 20 40 50 60 True 5 Overflow 5 10 20 40 50 50

Note:
You have to **implement** the stack operation functions **by yourself** for this task. Do **not** use the STL stack here.

Task 1(assignment): C++ offers a header file called **<stack>** which has different stack operations implemented as library functions. Follow this (https://www.geeksforgeeks.org/stack-in-cpp-stl/) and try to understand the usage of each function. Finally, implement **Task-1** using those STL library functions (assignment).

Task 2: Parsing HTML code with Stacks

HTML stands for Hyper Text Markup Language· HTML is the standard markup language for creating Web pages· It describes the structure of a Web page using different ‘**tags**’. Each tag has two portions - ‘opening’ and ‘closing’ with the notations‘ <tagName>’ and ‘</tagName>, respectively.

Given a snippet of HTML code, your task is to check whether the tags are properly nested or not.
Cases:

- ‘No error’: If all the tags are properly given.
- ‘Error at line ##’: If any error is detected.

Input	**Output**
8 <html> <head> <title> title of webpage </title> </head> <body> This is a paragraph </body> </html> -1	No error
8 <html> <head> <title> title of webpage </head> </title> <body> This is a paragraph </body> </html> -1	Error at Line 3
9 <html> <head> <title> </title> </head> <body> An unordered HTML list This is a paragraph </body> </html> -1	Error at line 6
9 <html> <head> <title> </title>	Error at line 7

</head> <body> An unordered HTML list This is a paragraph </body> </html> -1	
8 <html> <head> <title> </title> </head> An unordered HTML list This is a paragraph </body> </html> -1	Error at line 7
7 <html> <head> <title> </title> </head> <body> An unordered HTML list This is a paragraph -1	Error at line 7

Note:

- You just have to return the occurrence of the first error detected by your solution.
- For the last test case, there are some opening tags, for which no closing tags were found, although the entire code was checked. So we assume that there is an error in the last line.
- For simplicity, we’ve omitted the tags like
 for which no closing tags are defined.
- **You can use STL <stack> for this task.**

Task 3: Build an Array With Stack Operations

Suppose you are given a target array and an integer n. In each iteration, assume that you have n numbers available in a list (1,2,3,... n).
Your task is to build the target array using the following operations:

- "Push": Reads a new element from the beginning list and pushes it into the array.
- "Pop": Deletes the last element of the array.
- If the target array is already built, stop reading more elements.

Return a list of the operations needed to build the target array. If there are multiple valid answers, return any of them.

Input	**Output**	**Explanation**
1 3 3	Push Push Pop Push	Read number 1 and automatically push in the array -> [1] Read number 2 and automatically push in the array, then Pop it since 2 is not part of the target array -> [1] Read number 3 and automatically push in the array -> [1,3]
1 2 3 3	Push Push Push	Push 1, Push 2, Push 3
1 2 4	Push Push	Push 1, Push 2. No further operation is required.
2 3 4 4	Push Pop Push Push Push	Push 1, Pop 1, Push 2, Push 3, Push 4
1 4 4	Push Push Push Pop Pop Push **Alternative**: Push Push Pop Push Pop Push	Push 1, Push 2, Push 3, Pop 2, Pop 3, Push 4 Push 1, Push 2, Pop, Push 3, Pop, Push 4
1 3 4 6 6	Push Push Pop Push Push Push Pop Push	Push 1, Push 2, Pop, Push 3, Push 4, Push 5, Pop, Push 6

Note: You can use STL <stack> for this task.

Task-4: Next Greater Element

Given a set of numbers, your task is to print the ***Next Greater Element (NGE)*** for every element. The NGE for an element ***x*** is the first greater element on the right side of ***x*** in the array. Element for which no NGE exists, consider the NGE-value as -1.

Examples:

- 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 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.

Input:
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.)

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

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

Task 06: Checking parenthesis in Mathematical Expressions

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 (+ - * /) and any parenthesis ( )/[ ]/{ }.

The output will be Yes/No representing whether it is properly parenthesized.

Sample Input	Sample Output
10 [5 + (2 * 5) - (7 / 2)] [1 + { 3 * (2 / 3)] } [(1 + 1)] [(1 + 1]) [( )] { } { [( ) ( )] ( ) } ((([5 + (2 * 5) - (7 / 2) 5 + (2 * 5) - (7 / 2)] ( ))) ((( ))	 Yes No Yes No Yes No No No No No

Note: May need to use fflush(stdin) if you face problem to take input.

Task 07: Evaluating postfix notation

Write a program to 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

Note: You can use STL <stack> for this task.

Task 8: Parentheses checking in a Code

Write a program that will take a block of code consisting of N lines as input and return if there is any error in the code or not. Consider the ()[]{} symbols as valid parenthesis.

There will be three types of messages:

- ‘No Error’: If the code is compiled successfully.
- ‘Improper parenthesis’: If the code has the necessary number of parentheses but the wrong ones.
- ‘Missing parentheses’ If there is any shortage of parenthesis.

(For this task we only consider Error vs No Error)

<table>
<tr><th>Input</th><th>Output</th></tr>
<tr><td><pre>5
int main(){
  int x=5, arr[10];
  if(x == 5) return True;
  Else return false;
}</pre></td><td>No Errors.</td></tr>
<tr><td><pre>5
int main(){{
  int x=5, arr[10];
  if(x == 5) return True;
  Else return false;
}</pre></td><td>Error</td></tr>
<tr><td><pre>5
int main(){
  int x=5, arr[10];
  if(x == 5) return True;
  Else return false;
}}</pre></td><td>Error</td></tr>
<tr><td><pre>5
int main(){
  int x=5, arr[10);
  if(x == 5) return True;
  Else return false;
}</pre></td><td>Error</td></tr>
<tr><td><pre>7
for (i=0; i<10;i++){
  printf(“Hello %d”, i);
  if(i%2 == 0){
    print(“even”);
  else{
    print(“odd”);
  }
}</pre></td><td>Error</td></tr>
<tr><td>9</td><td>Error</td></tr>
</table>

```
for (i=0; i<10;i++)
  printf("Hello %d", i);
  if(i%2 == 0){
    print("even");
  }
  else{
    print("odd");
  }
}
```

Note: Use **cin.ignore**() after taking the number of line (sometimes the newline after N gets taken as part of input.

Limitation:
We have omitted some cases like,

```
for (i=0; i<10;i++)
  printf("Hello %d", i);
  if(i%2 == 0){
    print("even");
  else{
    print("odd");
  }
}
```

Where the closing curly brace at line-7 is a bit ambiguous. If we follow the trivial procedure, program will think that it is the closing bracket for the if condition and give no error. It is true in a sense. We are limited here with the ability to just check the number of parenthesis and not the correctness of the code. We can't also check that there is no parenthesis provided for the 'for loop'.

Task 9: Remove All Adjacent Duplicates In String

You are given a string *S* consisting of lowercase English letters. A duplicate removal consists of choosing two **adjacent** and **equal** letters and removing them. We repeatedly make duplicate removals on *S* until we no longer can.

Given the string, your task is to return the final string after all such duplicate removals have been made. (It can be proven that the answer is unique).

Input	**Output**	**Explanation**
abbaca	ca	‘abbaca’ we could remove ‘bb’ since the letters are adjacent and equal. The result of this move is that the string is ‘aaca’, of which only ‘aa’ has to be removed, so the final string is ‘ca’.
abbaac	ac	
abbbaaa	aba	
aaabaabaaa	Null	
cxyyyxccc	cxyxc	
geeksforgeeg	gksfor	
datastructure	datastructure	
caaabbbaacdddd	cabc	

<u>Task 10</u>: STPAR - Street Parade

For sure, the love mobiles will roll again on this summer's street parade. Each year, the organizers 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 organizers want them to be.

<u>Input</u>
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 on the approach street. No more than 1000 love mobiles participate in the street parade.

<u>Output</u>
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.

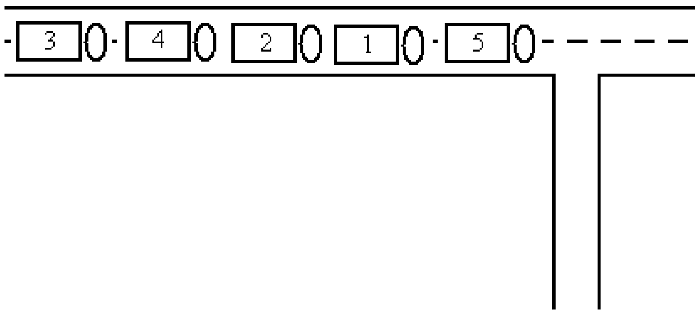
(PTO)

Input	Output
5 5 1 2 4 3	Yes
5 5 3 1 2 4	Yes
5 5 4 3 2 1	Yes
5 1 2 4 3 5	Yes
5 1 2 5 3 4	Yes
5 4 5 1 2 3	No
5 1 4 3 5 2	No
5 4 1 2 3 5	Yes
6 5 2 1 4 3 6	Yes
6 5 2 1 6 4 3	No
5 2 1 5 3 4	Yes
5 1 2 4 5 3	No

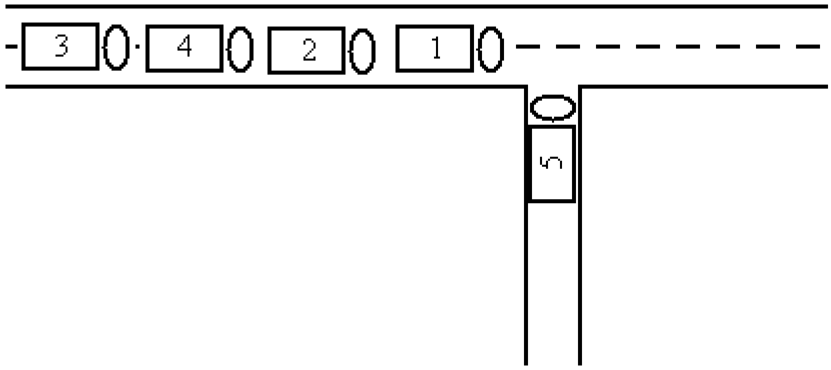
(Check the illustration for test case-1)

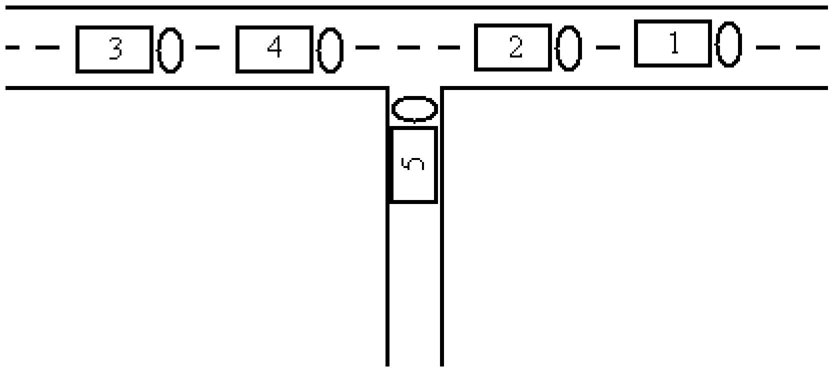
Illustration (for first test case):

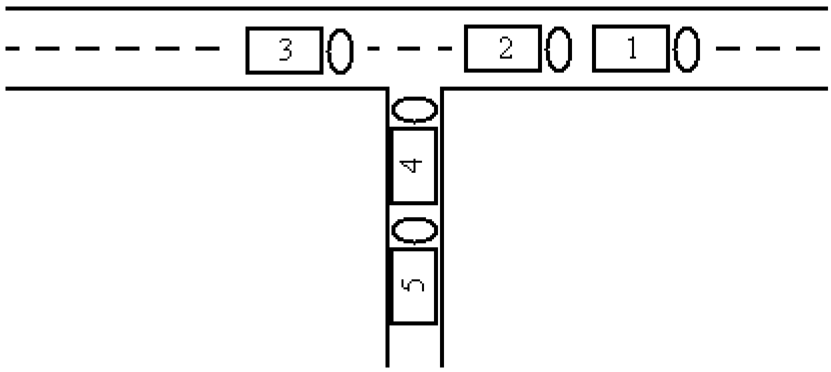
The sample input reflects the following situation:

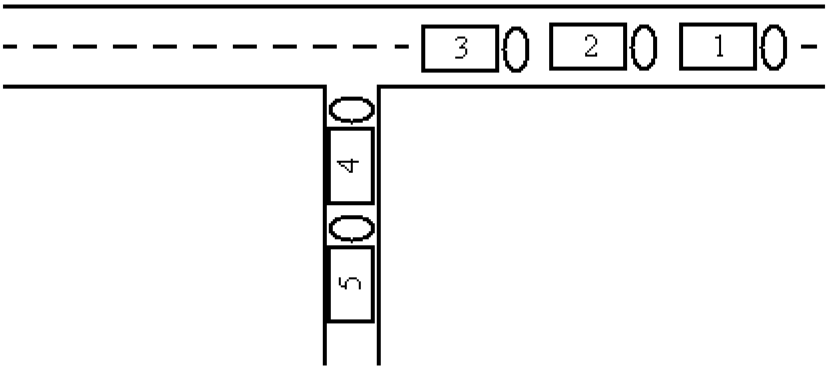


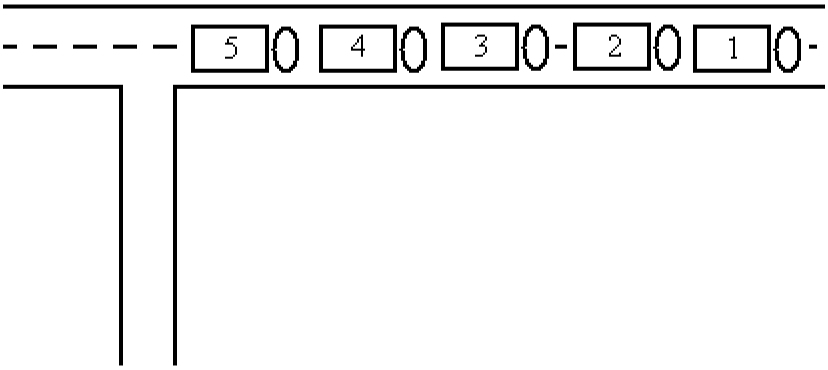
The five trucks can be re-ordered in the following way:








3
2
1
4
5


5
4
3
2
1

Task 11 – Colorful Shuttlecocks

Problem Statement

In order to play badminton with your friends, you bought a cylindrical container full of n Yonex Aerosensa-50 shuttlecocks. The shuttlecocks are numbered from top to bottom, i.e., the topmost shuttlecock is labeled with index 1, and the shuttlecock at the very bottom is labeled with the index n. The shuttlecocks have their own colors – the i-th shuttlecock has the color a_i.

You will be given q queries. For each of the j-th query t_j , you should –

- find the highest shuttlecock in the container with color t_j, i.e., the shuttlecock with the minimum index;
- print the position of the shuttlecock you found;
- take the shuttlecock and place it at the topmost position in the container.

Input

The first line contains two integers n and q – the number of shuttlecocks in the container and the number of queries.
The second line contains n integers a_1, a_2, ..., a_n – the colors of the shuttlecocks.
The third line contains q integers t_1, t_2 ..., t_q – the query colors.
It's guaranteed that queries ask only for colors that are present within the shuttlecock container.

Output

Print q integers – the answers for each query.

Sample Test Case(s)

Input

```
7 5
2 1 1 4 3 3 1
3 2 1 1 4
```

Output

```
5 2 3 1 5
```

Task 12 – Playing with Pringles

Problem Statement

One of the most popular brands of potato chips, *Pringles*, is known for its chips which have a distinctive saddle-shaped appearance. Suppose, you have n such potato chips, where n is an even number.

Each chip can be represented as a Left Parenthesis Shape '(' or a Right Parenthesis Shape ')'. As a perfectionist, you want to arrange the chips in a ***Perfect Sequence***.

A perfect sequence has the following properties –

- "()" is a perfect sequence of parenthesis.
- if S is a perfect sequence, then '(' + S + ')' is also a perfect sequence.
- if S_1 and S_2 are both perfect sequences, S_1 + S_2 is also a perfect sequence.

According to the properties, "(())", "()()", and "(()())" are perfect sequences, but ")(", "())(()", and "((" are not.
You decide to arrange the chips using minimum moves. Each move is defined as the act of moving exactly one chip from any position to either the start or the end of the sequence.
You are given the parenthesis sequence that the Pringles container had when you open it. Find how many moves you need to make it a perfect sequence of parenthesis.

Input

The first line contains one integer t – the number of test cases. Then t cases follow.
The first line of each case contains one even number n – the number of chips.
The second line of each case contains a string s consisting of an equal number of left and right parenthesis shapes - the sequence that the can of Pringles initially had.

Output

For each test case, print the minimum number of moves you need.

Sample Test Case(s)

Input

```
2
8
())()()(
10
)))((((())
```

Output

```
1
3
```

Task 13 – Regular Parenthesis Sequence

Problem Statement

A bracket sequence is considered regular if it can be transformed into a valid arithmetic expression by adding the characters “+” and “1” at appropriate positions within the sequence. For example, sequences “(())()”, “()”, and “(()(()))” are regular, while “)(”, “(()”, and “(()))(” are not.
Johnny found a bracket sequence and wanted to extract a regular bracket sequence by removing certain brackets. What’s the maximum length of the resulting regular bracket sequence?

Input

Input consists of a single line with non-empty string s of ‘(’ and ‘)’ characters.

Output

Output the maximum possible length of a regular bracket sequence.

Sample Test Case(s)

Input

```
(()))(
```

Output

```
4
```

Input

```
((()())
```

Output

```
6
```