You are given a pointer to the root of a binary search tree and values to be inserted into the tree. Insert the values into their appropriate position in the binary search tree and return the root of the updated binary tree. You just have to complete the function.

Input Format

You are given a function,

```
Node * insert (Node * root ,int data) {

}
```

Constraints

- No. of nodes in the tree ≤ 500

Output Format

Return the root of the binary search tree after inserting the value into the tree.

Sample Input

```
4
```

Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0

Input (stdin) Download

```
1 6
2 4 2 3 1 7 6
```

Your Output (stdout)

```
1 4 2 1 3 7 6
```

Expected Output Download

```
1 4 2 1 3 7 6
```

This challenge is part of a tutorial track by MyCodeSchool and is accompanied by a video lesson.

You are given the pointer to the head node of a linked list and an integer to add to the list. Create a new node with the given integer. Insert this node at the tail of the linked list and return the head node of the linked list formed after inserting this new node. The given head pointer may be null, meaning that the initial list is empty.

Function Description

Complete the *insertNodeAtTail* function with the following parameters:

- *SinglyLinkedListNode pointer head*: a reference to the head of a list
- *int data*: the data value for the node to insert

Returns

- *SinglyLinkedListNode pointer*: reference to the head of the modified linked list

Input Format

The first line contains an integer n, the number of elements in the linked list. The next n lines contain an integer each, the value that needs to be inserted at tail.

Constraints

Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0

Sample Test case 1

Your Output (stdout)

```
141
302
164
530
474
```

Expected Output Download

```
141
302
164
530
474
```

Problem

Complete the *postOrder* function in the editor below. It received 1 parameter: a pointer to the root of a binary tree. It must print the values in the tree's postorder traversal as a single line of space-separated values.

Input Format

Our test code passes the root node of a binary tree to the *postOrder* function.

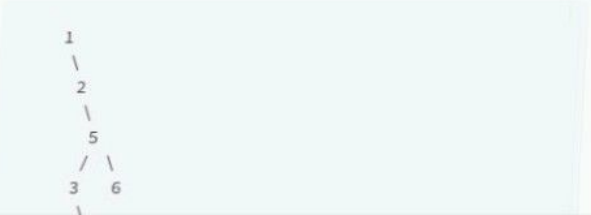
Constraints

$1 \leq$ Nodes in the tree ≤ 500

Output Format

Print the tree's postorder traversal as a single line of space-separated values.

Sample Input



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0

Sample Test case 1

Input (stdin) Download

```
1 6
2 1 2 5 3 6 4
```

Your Output (stdout)

```
1 4 3 6 5 2 1
```

Expected Output Download

```
1 4 3 6 5 2 1
```

A $left\ rotation$ operation on a circular array shifts each of the array's elements 1 unit to the left. The elements that fall off the left end reappear at the right end. Given an integer d, rotate the array that many steps to the left and return the result.

Example

$d = 2$

$arr = [1, 2, 3, 4, 5]$

After 2 rotations, $arr' = [3, 4, 5, 1, 2]$.

Function Description

Complete the $rotateLeft$ function with the following parameters:

- $int\ d$: the amount to rotate by
- $int\ arr[n]$: the array to rotate

Returns

- $int[n]$: the rotated array

Input Format

The first line contains two space-separated integers that denote n, the number of integers, and d, the number of left rotations to perform.

The second line contains n space-separated integers that describe $arr[]$.

Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0

Input (stdin) Download

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

Your Output (stdout)

```
5 1 2 3 4
```

Expected Output Download

```
5 1 2 3 4
```

A bracket is considered to be any one of the following characters: (,), {, }, [, or].

Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e.,),], or }) of the exact same type. There are three types of matched pairs of brackets: [], {}, and ().

A matching pair of brackets is not balanced if the set of brackets it encloses are not matched. For example, {[(])} is not balanced because the contents in between { and } are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, (, and the pair of parentheses encloses a single, unbalanced closing square bracket,].

By this logic, we say a sequence of brackets is balanced if the following conditions are met:

- It contains no unmatched brackets.
- The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Given n strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, return YES. Otherwise, return NO.

Function Description

Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0

Sample Test case 1

Sample Test case 2

```
2 {[()]}
3 {[(])}
4 {{[[(())]]}}
```

Your Output (stdout)

```
1 YES
2 NO
3 YES
```

Expected Output Download

```
1 YES
2 NO
3 YES
```