

# RAMAIAH INSTITUTE OF TECHNOLOGY

MSRIT NAGAR, BENGALURU, 560054



LeetCode

## [Data Structures Lab]

*Submitted in partial fulfilment of the OTHER COMPONENT requirements as a part of the Data Structures Lab with code ISL36 for the III Semester of degree of **Bachelor of Engineering in Information Science and Engineering***

Submitted by

**Candidate Name**

**1MS22IS139**

**SUHAS S**

Under the Guidance of

**Faculty Incharge**

**SHIVANANDA S**

**Department of Information Science and Engineering**

**Ramaiah Institute of Technology**

2023 – 2024

# 1. Design a Stack With Increment Operator

The screenshot shows a coding platform interface. On the left, the problem description for '1381. Design a Stack With Increment Operation' is displayed. It includes the problem statement, constraints, and an example. On the right, the C++ solution code is shown, implementing the CustomStack class with methods for push, pop, and increment operations.

**1381. Design a Stack With Increment Operation** Solved

Medium Topics Companies Hint

Design a stack that supports increment operations on its elements.

Implement the `CustomStack` class:

- `CustomStack(int maxSize)` Initializes the object with `maxSize` which is the maximum number of elements in the stack.
- `void push(int x)` Adds `x` to the top of the stack if the stack has not reached the `maxSize`.
- `int pop()` Pops and returns the top of the stack or `-1` if the stack is empty.
- `void inc(int k, int val)` Increments the bottom `k` elements of the stack by `val`. If there are less than `k` elements in the stack, increment all the elements in the stack.

**Example 1:**

**Input**  
["CustomStack", "push", "push", "pop", "push", "push", "push", "increment", "increment", "pop", "pop", "pop", "pop"]  
[[3], [1], [2], [1], [2], [3], [4], [5, 100], [2, 100], [1], [1], [1], [1]]

**Output**  
[null, null, null, 2, null, null, null, null, null, 103, 202, 201, -1]

**Explanation**  
CustomStack stk = new CustomStack(3); // Stack is Empty []  
stk.push(1);

```
1 typedef struct {
2     int *arr;
3     int ms;
4     int top;
5 } CustomStack;
6
7
8 CustomStack* customStackCreate(int maxSize) {
9     CustomStack *stack = malloc(sizeof(CustomStack));
10    stack->arr = (int*) malloc(sizeof(int)*maxSize);
11    stack->ms = maxSize;
12    stack->top = -1;
13    return stack;
14 }
15
16
17 void customStackPush(CustomStack* obj, int x) {
18     if(obj->top == obj->ms - 1)
```

Saved to local Ln 38, Col 2

Testcase Test Result

Case 1 +

["CustomStack", "push", "push", "pop", "push", "push", "push", "increment", "increment", "pop", "pop", "pop", "pop"]

Source

```
typedef struct {
    int *arr;
    int ms;
    int top;
} CustomStack;

CustomStack* customStackCreate(int maxSize) {
    CustomStack *stack = malloc(sizeof(CustomStack));
    stack->arr = (int*) malloc(sizeof(int)*maxSize);
    stack->ms = maxSize;
    stack->top = -1;
    return stack;
}

void customStackPush(CustomStack* obj, int x) {
    if(obj->top == obj->ms - 1)
        return;
    obj->arr[++(obj->top)] = x;
}

int customStackPop(CustomStack* obj) {
    if(obj->top == -1)
        return -1;
    return obj->arr[(obj->top)--];
}

void customStackIncrement(CustomStack* obj, int k, int val) {
    if(k >= obj->ms)
        k = obj->ms;
    for(int i = 0; i < k; i++)
        obj->arr[i] += val;
}

void customStackFree(CustomStack* obj) {
}
```

## 2. Valid Parentheses

**20. Valid Parentheses** Solved

Easy Topics Companies Hint

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['`, and `']'`, determine if the input string is valid.

An input string is valid if:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.
- Every close bracket has a corresponding open bracket of the same type.

**Example 1:**  
Input: `s = "()"`  
Output: `true`

**Example 2:**  
Input: `s = "()[]{}"`  
Output: `true`

**Example 3:**  
Input: `s = "(]"`  
Output: `false`

23.3K 298 1 0 0

```
1 bool isValid(char* s)
2 {
3     int top=-1;
4     char stack[10000];
5
6     for(int i=0;s[i] != '\0';++i)
7     {
8         char c = s[i];
9         if(c == '(' || c == '{' || c == '[')
10        {
11            stack[++top] = c;
12        }
13        else if (c == ')' || c == '}' || c == ']')
14        {
15            if (top == -1) {
16                return false;
17            }
18            char opening = stack[top--];
19            if (opening != '(' && c == ')') {
20                return false;
21            } else if (opening != '{' && c == '}') {
22                return false;
23            } else if (opening != '[' && c == ']') {
24                return false;
25            }
26        }
27    }
28    return top == -1;
29 }
```

Restored from local Upgrade to Cloud Saving Ln 1, Col 1

Testcase Test Result

Case 1 Case 2 Case 3 +

s =

Source

```
bool isValid(char* s)
{
    int top=-1;
    char stack[10000];

    for(int i=0;s[i] != '\0';++i)
    {
        char c = s[i];
        if(c == '(' || c == '{' || c == '[')
        {
            stack[++top] = c;
        }
        else if (c == ')' || c == '}' || c == ']')
        {
            if (top == -1) {
                return false;
            }
            char opening = stack[top--];
            if (opening != '(' && c == ')') {
                return false;
            } else if (opening != '{' && c == '}') {
                return false;
            } else if (opening != '[' && c == ']') {
                return false;
            }
        }
    }
    return top == -1;
}
```

### 3. Pow(x,n)

The screenshot shows the LeetCode interface for the problem "50. Pow(x, n)". The problem is marked as "Solved" and "Medium". The description states: "Implement `pow(x, n)`, which calculates  $x$  raised to the power  $n$  (i.e.,  $x^n$ )." Examples provided are: Example 1: Input:  $x = 2.00000$ ,  $n = 10$ , Output:  $1024.00000$ ; Example 2: Input:  $x = 2.10000$ ,  $n = 3$ , Output:  $9.26100$ ; Example 3: Input:  $x = 2.00000$ ,  $n = -2$ , Output:  $0.25000$ , Explanation:  $2^{-2} = 1/2^2 = 1/4 = 0.25$ . Constraints:  $-100.0 < x < 100.0$ ,  $-2^{31} \leq n \leq 2^{31} - 1$ . The C++ code on the right implements a recursive function `myPow` that handles base cases for  $n=0$  and  $n=INT\_MIN$ , and uses a divide-and-conquer approach for negative  $n$  and recursive calls for  $n/2$ .

```
#include <stdio.h>
#include <limits.h> // for INT_MIN definition

double myPow(double x, int n) {
    if (n == 0) {
        return 1.0;
    }

    if (n == INT_MIN) {
        return myPow(x, n + 1) / x;
    }

    if (n < 0) {
        x = 1.0 / x;
        n = -n;
    }

    double halfPower = myPow(x, n / 2);

    if (n % 2 == 0) {
        return halfPower * halfPower;
    } else {
        return halfPower * halfPower * x;
    }
}
```

```
#include <stdio.h>
#include <limits.h> // for INT_MIN definition

double myPow(double x, int n) {
    if (n == 0) {
        return 1.0;
    }

    if (n == INT_MIN) {
        return myPow(x, n + 1) / x;
    }

    if (n < 0) {
        x = 1.0 / x;
        n = -n;
    }

    double halfPower = myPow(x, n / 2);

    if (n % 2 == 0) {
        return halfPower * halfPower;
    } else {
        return halfPower * halfPower * x;
    }
}
```

## 4. Swap Nodes In Paris

**24. Swap Nodes in Pairs** Solved

Medium Topics Companies

Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed).

**Example 1:**

```
graph LR; 1((1)) --> 2((2)); 2 --> 3((3)); 3 --> 4((4));
```

Input: head = [1,2,3,4]  
Output: [2,1,4,3]

**Example 2:**

Input: head = []  
Output: []

11.7K 66

**Code**

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     struct ListNode *next;
6  * };
7  */
8 struct ListNode* cur;
9 struct ListNode* swap(struct ListNode* head)
10 {
11     struct ListNode* temp;
12     if(head==NULL)
13         return head;
14     else{
15         if(head->next!=NULL){
16             temp=head->next;
17             head->next=temp->next;
18             temp->next=head;
19             cur=swap(head->next);
20             head->next=cur;
21         }
22         else
23             return head;
24     }
25 }
```

Saved to local Ln 33, Col 2

Testcase Test Result

Case 1 Case 2 Case 3 +

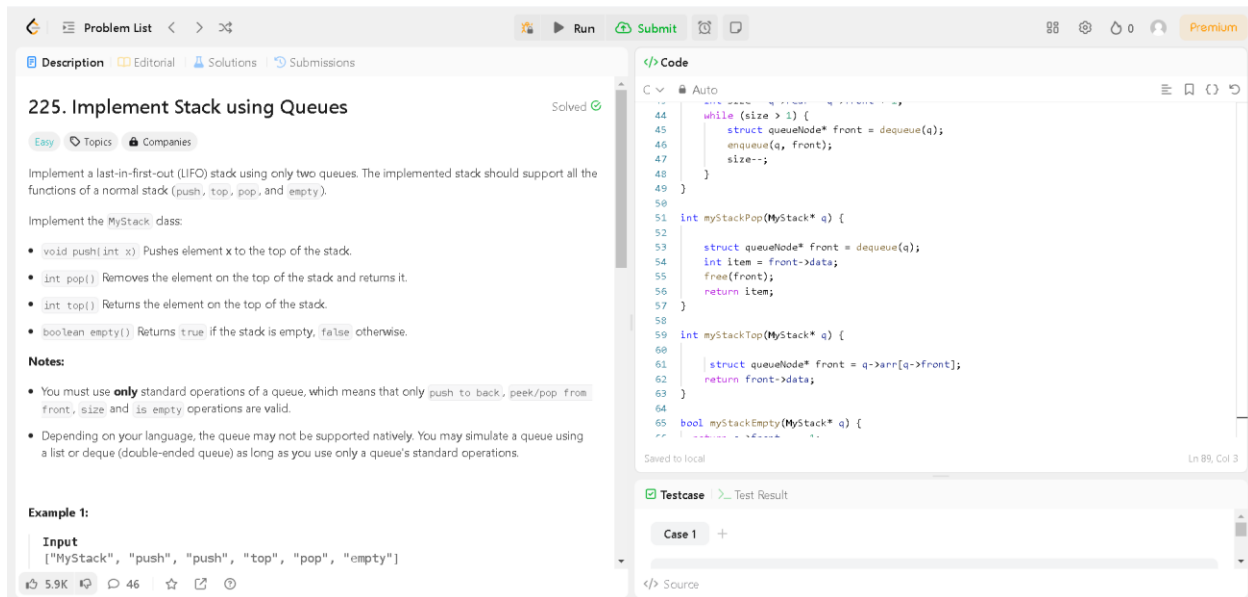
head =

</> Source

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* cur;
struct ListNode* swap(struct ListNode* head)
{
    struct ListNode* temp;
    if(head==NULL)
        return head;
    else{
        if(head->next!=NULL){
            temp=head->next;
            head->next=temp->next;
            temp->next=head;
            cur=swap(head->next);
            head->next=cur;
        }
        else
            return head;
    }
    return temp;
}
struct ListNode* swapPairs(struct ListNode* head) {
    if(head==NULL || head->next==NULL)
        return head;

    head=swap(head);
    return head;
}
```

## 5. Implement Stack Using Queues



**225. Implement Stack using Queues** Solved

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

Implement the `MyStack` class:

- `void push(int x)` Pushes element `x` to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

**Notes:**

- You must use **only** standard operations of a queue, which means that only `push to back`, `peek/pop from front`, `size` and `is empty` operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

**Example 1:**

**Input**  
["MyStack", "push", "push", "top", "pop", "empty"]

**Code**

```
44 while (size > 1) {
45     struct queueNode* front = dequeue(q);
46     enqueue(q, front);
47     size--;
48 }
49
50
51 int myStackPop(MyStack* q) {
52
53     struct queueNode* front = dequeue(q);
54     int item = front->data;
55     free(front);
56     return item;
57 }
58
59 int myStackTop(MyStack* q) {
60
61     struct queueNode* front = q->arr[q->front];
62     return front->data;
63 }
64
65 bool myStackEmpty(MyStack* q) {
66     // return q->front == -1;
67 }
```

```
struct queueNode{
    int data;
};

typedef struct {
    struct queueNode *arr[101];
    int front;
    int rear;
} MyStack;

MyStack* myStackCreate() {
    MyStack *q =(MyStack *)malloc(sizeof(MyStack));
    q->front=-1;
    q->rear=-1;
    return q;
}

void enqueue(MyStack* q, struct queueNode* item) {
    q->arr[++q->rear] = item;
    if (q->front == -1) {
        q->front = 0;
    }
}

struct queueNode* dequeue(MyStack* q) {
    struct queueNode* item = q->arr[q->front];
    if (q->front == q->rear) {
        q->front = q->rear = -1;
    } else {
        q->front++;
    }
    return item;
}

void myStackPush(MyStack* q, int x) {
    struct queueNode* node = (struct queueNode*) malloc(sizeof(struct queueNode));
    node->data = x;
    enqueue(q, node);
    int size = q->rear - q->front + 1;
    while (size > 1) {
        struct queueNode* front = dequeue(q);
        enqueue(q, front);
        size--;
    }
}

int myStackPop(MyStack* q) {
    struct queueNode* front = dequeue(q);
    int item = front->data;
    free(front);
    return item;
}

int myStackTop(MyStack* q) {
    struct queueNode* front = q->arr[q->front];
    return front->data;
}

bool myStackEmpty(MyStack* q) {
    return q->front == -1;
}

void myStackFree(MyStack* q) {
    while (!myStackEmpty(q)) {
        struct queueNode* front = dequeue(q);
        free(front);
    }
    free(q);
}
```

## 6. Design Circular Queue

**622. Design Circular Queue** Solved

Medium Topics Companies

Design your implementation of the circular queue. The circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle, and the last position is connected back to the first position to make a circle. It is also called "Ring Buffer".

One of the benefits of the circular queue is that we can make use of the spaces in front of the queue. In a normal queue, once the queue becomes full, we cannot insert the next element even if there is a space in front of the queue. But using the circular queue, we can use the space to store new values.

Implement the `MyCircularQueue` class:

- `MyCircularQueue(k)` Initializes the object with the size of the queue to be `k`.
- `int Front()` Gets the front item from the queue. If the queue is empty, return `-1`.
- `int Rear()` Gets the last item from the queue. If the queue is empty, return `-1`.
- `boolean enqueue(int value)` Inserts an element into the circular queue. Return `true` if the operation is successful.
- `boolean dequeue()` Deletes an element from the circular queue. Return `true` if the operation is successful.
- `boolean isEmpty()` Checks whether the circular queue is empty or not.
- `boolean isFull()` Checks whether the circular queue is full or not.

You must solve the problem without using the built-in queue data structure in your programming language.

3.5K 20 5 0 0

**Code**

```
1 typedef struct
2 {
3     int front;
4     int rear;
5     int *a;
6     int n;
7 } MyCircularQueue;
8
9 MyCircularQueue* myCircularQueueCreate(int k)
10 {
11     MyCircularQueue* q = malloc(sizeof *q);
12     q->a = (int*) malloc(sizeof(int) * k);
13     q->front = -1;
14     q->rear = -1;
15     q->n = k;
16     return q;
17 }
18
19
20 bool myCircularQueueEnqueue(MyCircularQueue* q, int value)
21 {
22     if((q->rear + 1) % q->n == q->front)
```

Saved to local Ln 88, Col 2

Testcase Test Result

Case 1 +

Source

```
typedef struct
{
    int front;
    int rear;
    int *a;
    int n;
} MyCircularQueue;

MyCircularQueue* myCircularQueueCreate(int k)
{
    MyCircularQueue* q = malloc(sizeof *q);
    q->a = (int*) malloc(sizeof(int) * k);
    q->front = -1;
    q->rear = -1;
    q->n = k;
    return q;
}

bool myCircularQueueEnqueue(MyCircularQueue* q, int value)
{
    if((q->rear + 1) % q->n == q->front)
        return false;
    else
    {
        if (q->front == -1)
            q->front = 0;
        q->rear = (q->rear + 1) % q->n;
        q->a[q->rear] = value;
        return true;
    }
}

bool myCircularQueueDequeue(MyCircularQueue* q)
{
    if(q->front == -1)
        return false;
    else
    {
        if (q->front == q->rear)
        {
            q->front = -1;
            q->rear = -1;
        }
        else
            q->front = (q->front + 1) % q->n;
        return true;
    }
}
```

```
int myCircularQueueFront(MyCircularQueue* q)
{
    if(q->front == -1)
        return -1;
    return q->a[q->front];
}

int myCircularQueueRear(MyCircularQueue* q)
{
    if(q->front == -1)
        return -1;
    return (q->a[q->rear]);
}

bool myCircularQueueIsEmpty(MyCircularQueue* q)
{
    if(q->front == -1)
        return true;
    return false;
}

bool myCircularQueueIsFull(MyCircularQueue* q)
{
    if((q->rear + 1) % q->n == q->front)
        return true;
    return false;
}

void myCircularQueueFree(MyCircularQueue* obj)
{
    free(obj);
}
```

## 7. Linked List Cycle

Problem List

Run

Submit

Premium

Description

Editorial

Solutions

Submissions

### 141. Linked List Cycle

Solved

Easy

Topics

Companies

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter**.

Return `true` if there is a cycle in the linked list. Otherwise, return `false`.

**Example 1:**

**Input:** `head = [3,2,0,-4]`, `pos = 1`  
**Output:** `true`  
**Explanation:** There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

**Example 2:**

Code

C

Auto

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     struct ListNode *next;
6  * };
7  */
8 bool hasCycle(struct ListNode *head) {
9     struct ListNode* a = head,*b=head;
10    while(a!= NULL && a->next != NULL){
11        b = b->next;
12        a = a->next->next;
13        if(a == b)
14            return true;
15    }
16    return false;
17 }
```

Saved to local

Ln 17, Col 2

Testcase

Test Result

Case 1	Case 2	Case 3	+
[3,2,0,-4]			
1			

</> Source

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
bool hasCycle(struct ListNode *head) {
    struct ListNode* a = head,*b=head;
    while(a!= NULL && a->next != NULL){
        b = b->next;
        a = a->next->next;
        if(a == b)
            return true;
    }
    return false;
}
```



## 8. Odd Even Linked List

Problem List < > 🔍

Run Submit ⌛

88 0 Premium

Description Editorial Solutions Submissions

### 328. Odd Even Linked List

Solved ✓

Medium Topics Companies

Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the *reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in  $O(1)$  extra space complexity and  $O(n)$  time complexity.

**Example 1:**

**Input:** `head = [1,2,3,4,5]`  
**Output:** `[1,3,5,2,4]`

**Example 2:**

</> Code

C Auto

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     struct ListNode *next;
6  * };
7  */
8 struct ListNode* oddEvenList(struct ListNode* head) {
9     if(head==NULL || head->next==NULL)
10         return head;
11     struct ListNode *odd,*even,*evenHead;
12     odd=head;
13     even=head->next;
14     evenHead=even;
15
16     while(even!=NULL && even->next!=NULL)
17     {
18         odd->next=odd->next->next;
19         even->next=even->next->next;
20         even=even->next;
```

Saved to local Ln 25, Col 2

✓ Testcase > Test Result

Case 1 Case 2 +

head =

[1,2,3,4,5]

</> Source

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* oddEvenList(struct ListNode* head) {
    if(head==NULL || head->next==NULL)
        return head;
    struct ListNode *odd,*even,*evenHead;
    odd=head;
    even=head->next;
    evenHead=even;

    while(even!=NULL && even->next!=NULL)
    {
        odd->next=odd->next->next;
        even->next=even->next->next;
        even=even->next;
        odd=odd->next;
    }
    odd->next=evenHead;
    return head;
}
```

## 9. Same Tree

Problem List

Run

Submit

0

Premium

Description

Editorial

Solutions

Submissions

### 100. Same Tree

Solved

Easy

Topics

Companies

Given the roots of two binary trees  $p$  and  $q$ , write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

**Example 1:**

```
graph TD; p1((1)) --- p2((2)); p1 --- p3((3)); q1((1)) --- q2((2)); q1 --- q3((3));
```

**Input:**  $p = [1,2,3]$ ,  $q = [1,2,3]$   
**Output:** true

**Example 2:**

```
graph TD; p1((1)) --- p2((1)); q1((1)) --- q2((1));
```

11K 96

Code

C

Auto

```
1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     struct TreeNode *left;
6  *     struct TreeNode *right;
7  * };
8  */
9 bool isSameTree(struct TreeNode* p, struct TreeNode* q){
10     // base case: when one tree ends other tree should end too
11     if(p == NULL || q == NULL) return (p == q);
12
13     // otherwise, the values should be equal at each node level.
14     return (p->val == q->val && isSameTree(p->left, q->left) && isSameTree(p->right, q->right));
15 }
```

Saved to local

Ln 15, Col 2

Testcase

Test Result

Case 1

Case 2

Case 3

+

0 =

Source

```
bool isSameTree(struct TreeNode* p, struct TreeNode* q){
    // base case: when one tree ends other tree should end too
    if(p == NULL || q == NULL) return (p == q);

    // otherwise, the values should be equal at each node level.
    return (p->val == q->val && isSameTree(p->left, q->left) && isSameTree(p->right, q->right));
}
```

## 10. Validate Binary Search Tree

Problem List

98. Validate Binary Search Tree

Medium

Topics

Companies

Given the `root` of a binary tree, determine if it is a valid binary search tree (BST).

A **valid BST** is defined as follows:

- The left **subtree** of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

**Example 1:**

```
graph TD; 2((2)) --- 1((1)); 2 --- 3((3));
```

**Input:** `root = [2,1,3]`  
**Output:** `true`

Solved

```
1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     struct TreeNode *left;
6  *     struct TreeNode *right;
7  * };
8  */
9
10
11 bool help(struct TreeNode *root, long min, long max) {
12
13     if (root == NULL)
14         return true;
15
16     if (root->val <= min || root->val >= max)
17         return false;
18
19     if (!help(root->left, min, root->val) || !help(root->right, root->val, max))
20         return false;
21
22     return true;
23 }
```

Testcase Test Result

Case 1 Case 2 +

root =

Source

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */

bool help(struct TreeNode *root, long min, long max) {

    if (root == NULL)
        return true;

    if (root->val <= min || root->val >= max)
        return false;

    if (!help(root->left, min, root->val) || !help(root->right, root->val, max))
        return false;

    return true;
}

bool isValidBST(struct TreeNode* root) {

    if (root == NULL || (root->left == NULL && root->right == NULL))
        return true;

    return help(root, LONG_MIN, LONG_MAX);
}
```

## REAL TIME PROBLEM: Restaurant Menu Manager

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct MenuItem {
    char name[50];
    float price;
    struct MenuItem* next;
};

void displayMenu(struct MenuItem* head) {
    printf("Menu:\n");
    printf("-----\n");
    while (head != NULL) {
        printf("%s - $%.2f\n", head->name, head->price);
        head = head->next;
    }
    printf("-----\n");
}

struct MenuItem* addItem(struct MenuItem* head, char name[], float price)
{
    struct MenuItem* newItem = (struct MenuItem*)malloc(sizeof(struct
MenuItem));
    if (newItem == NULL) {
        printf("Error: Memory allocation failed.\n");
        return head;
    }

    strncpy(newItem->name, name, sizeof(newItem->name));
    newItem->price = price;
    newItem->next = head;
    printf("Item added: %s - $%.2f\n", newItem->name, newItem->price);

    return newItem;
}

struct MenuItem* deleteItem(struct MenuItem* head, char name[]) {
    struct MenuItem* current = head;
```

```

    struct MenuItem* prev = NULL;

    while (current != NULL && strcmp(current->name, name) != 0) {
        prev = current;
        current = current->next;
    }

    if (current == NULL) {
        printf("Error: Item not found.\n");
        return head;
    }

    if (prev == NULL) {
        head = current->next;
    } else {
        prev->next = current->next;
    }

    printf("Item deleted: %s - $%.2f\n", current->name, current->price);
    free(current);

    return head;
}

void freeMenu(struct MenuItem* head) {
    struct MenuItem* current = head;
    while (current != NULL) {
        struct MenuItem* next = current->next;
        free(current);
        current = next;
    }
}

float calculateTotal(struct MenuItem* order) {
    float total = 0.0;
    while (order != NULL) {
        total += order->price;
        order = order->next;
    }
    return total;
}

int main() {

```

```

    struct MenuItem* menu = NULL;
    struct MenuItem* order = NULL;

    menu = addItem(menu, "Burger", 5.99);
    menu = addItem(menu, "Pizza", 8.49);
    menu = addItem(menu, "Salad", 4.99);

    displayMenu(menu);

    order = addItem(order, "Burger", 5.99);
    order = addItem(order, "Pizza", 8.49);

    printf("\nOrder Summary:\n");
    displayMenu(order);

    float totalBill = calculateTotal(order);
    printf("\nTotal Bill: $%.2f\n", totalBill);

    // Cleaning up memory
    freeMenu(menu);
    freeMenu(order);

    return 0;
}

```

## Output:

```

PS D:\Myfiles\Msrit\Notes\Sem-3\data_structures\Leetcode programs> gcc Restauran_Menu_Manager.c
PS D:\Myfiles\Msrit\Notes\Sem-3\data_structures\Leetcode programs> ./a.exe
Item added: Burger - $5.99
Item added: Pizza - $8.49
Item added: Salad - $4.99
Menu:
-----
Salad - $4.99
Pizza - $8.49
Burger - $5.99
-----
Item added: Burger - $5.99
Item added: Pizza - $8.49

Order Summary:
Menu:
-----
Pizza - $8.49
Burger - $5.99
-----

Total Bill: $14.48
PS D:\Myfiles\Msrit\Notes\Sem-3\data_structures\Leetcode programs>

```