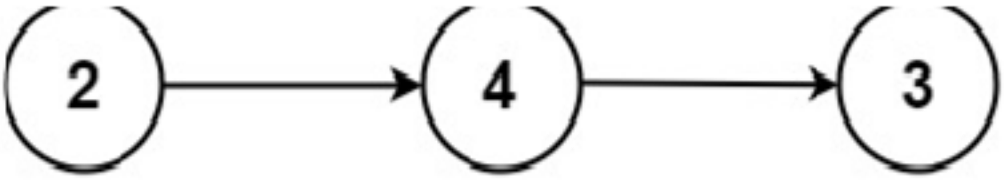
Write a program to create and display a linked list

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

struct ListNode {
    int val;
    struct ListNode* next;
};

struct ListNode* createNode(int val) {
    struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
    newNode->val = val;
    newNode->next = NULL;
    return newNode;
}

struct ListNode* insertEnd(struct ListNode* head, int val) {
    struct ListNode* newNode = createNode(val);
    if (head == NULL) {
        head = newNode;
    } else {
        struct ListNode* current = head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
    return head;
}

void displayList(struct ListNode* head) {
    struct ListNode* current = head;
    while (current != NULL) {
        printf("%d", current->val);
        if (current->next != NULL) {
            printf("->");
        }
        current = current->next;
    }
    printf("\n");
}

int main() {

    struct ListNode* head = NULL;
    head = insertEnd(head, 6);
    head = insertEnd(head, 7);
    head = insertEnd(head, 8);
    head = insertEnd(head, 9);

    printf("Output: ");
    displayList(head);

    return 0;
}
```

You are given with the following linked list



The digits are stored in the above order, you are asked to print the list in reverse order.

```
#include <stdio.h>
#include <stdlib.h>
struct ListNode
{
    int val;
    struct ListNode *next;
};

int main()
{
    struct ListNode *head = (struct ListNode *)malloc(sizeof(struct ListNode));
    head->val = 2;
    head->next = (struct ListNode *)malloc(sizeof(struct ListNode));
    head->next->val = 3;
    head->next->next = (struct ListNode *)malloc(sizeof(struct ListNode));
    head->next->next->val = 4;
    head->next->next->next = NULL;
    printf("Reversed Linked List: ");
    struct ListNode *prev = NULL;
    struct ListNode *cur = head;
    while (cur != NULL)
    {
        struct ListNode *next = cur->next;
        cur->next = prev;
        prev = cur;
        cur = next;
    }
    cur = prev;
    while (cur != NULL)
    {
        printf("%d ", cur->val);
        struct ListNode *temp = cur;
        cur = cur->next;
        free(temp);
```

```
    }
  return 0;
}
```

Given the head of a singly linked list, return number of nodes present in a linked

Example 1:
1->2->3->5->8
Output 5

```
#include <stdio.h>
#include <stdlib.h>
struct ListNode
{
    int val;
    struct ListNode *next;
};
struct ListNode* createNode(int val)
{
    struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
    if (!newNode)
    {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->val = val;
    newNode->next = NULL;
    return newNode;
}
int countNodes(struct ListNode* head)
{
    int count = 0;
    struct ListNode* current = head;
    while (current != NULL)
    {
        count++;
        current = current->next;
    }
    return count;
}
int main()
{
    struct ListNode* head = NULL;
    head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(3);
    head->next->next->next = createNode(4);
    int nodeCount = countNodes(head);
    printf("Number of nodes in the linked list: %d\n", nodeCount);
    return 0;
}
```

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the sum of these two arrays
Example 1:
Input: nums1 = [1,3], nums2 = [2]
Output: 6
Example 2:
Input: nums1 = [1,2], nums2 = [3,4]
Output: 10

```
#include <stdio.h>

int mergeAndSum(int nums1[], int m, int nums2[], int n) {
    int merged[m + n];
    int i = 0, j = 0, k = 0;
    int sum = 0;

    while (i < m && j < n) {
        if (nums1[i] < nums2[j])
            merged[k++] = nums1[i++];
        else
            merged[k++] = nums2[j++];
    }

    while (i < m)
        merged[k++] = nums1[i++];

    while (j < n)
        merged[k++] = nums2[j++];

    for (int l = 0; l < m + n; l++)
        sum += merged[l];

    return sum;
}

int main()
{
    int nums1[] = {1, 3};
    int nums2[] = {2};
    int m = sizeof(nums1) / sizeof(nums1[0]);
    int n = sizeof(nums2) / sizeof(nums2[0]);
    printf("%d\n", mergeAndSum(nums1, m, nums2, n));
    int nums3[] = {1, 2, 4};
    int nums4[] = {1, 3, 4};
    int p = sizeof(nums3) / sizeof(nums3[0]);
    int q = sizeof(nums4) / sizeof(nums4[0]);
    printf("%d\n", mergeAndSum(nums3, p, nums4, q));
    return 0;
}
```

You have been given a positive integer N. You need to find and print the
Factorial of this number without using recursion. The Factorial of a positive
integer N refers to the product of all number in the range from 1 to N.

```
#include <stdio.h>

unsigned long long factorial(int n) {
    unsigned long long result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

int main() {
    int N;
    printf("Enter a positive integer: ");
    scanf("%d", &N);

    if (N < 0) {
        printf("Factorial is not defined for negative numbers.\n");
    } else {
        unsigned long long fact = factorial(N);
        printf("Factorial of %d is %llu\n", N, fact);
    }

    return 0;
}
```

Find the factorial of a number using iterative procedure

```
#include <stdio.h>
unsigned long long factorial(int n)
{
    unsigned long long result = 1;
    for (int i = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}

int main()
{
    int N;
    printf("Enter a positive integer: ");
    scanf("%d", &N);

    if (N < 0)
```

```
    {
        printf("Factorial is not defined for negative numbers.\n");
    } else
    {
        unsigned long long fact = factorial(N);
        printf("Factorial of %d is %llu\n", N, fact);
    }

    return 0;
}
```

Given the head of a linked list, insert the node in nth place and return its head.

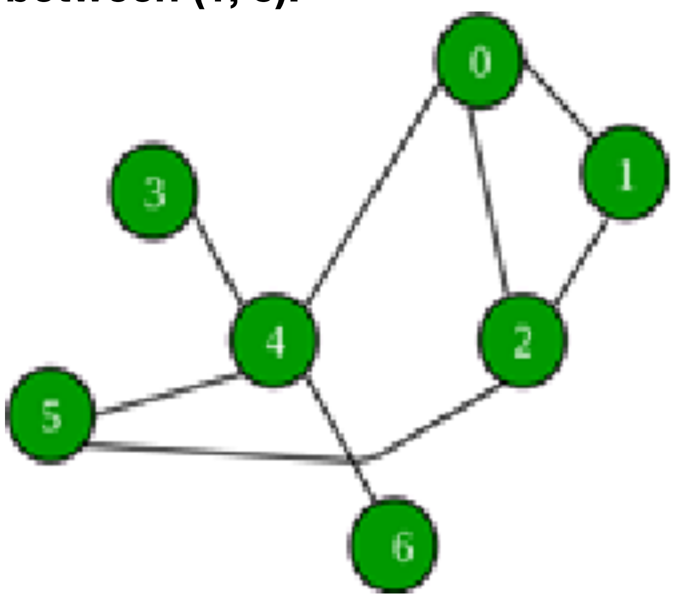
Input: head = [1,3,2,3,4,5], p=3 n = 2
Output: [1,3,2,3,4,5]
Input: head = [1], p = 0, n = 1
Output: [0,1]
Input: head = [1,2], p=3, n = 3
Output: [1,2,3]

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

struct ListNode {
    int val;
    struct ListNode *next;
};

struct ListNode* insertNode(struct ListNode* head, int p, int n) {
    struct ListNode *newNode = malloc(sizeof(struct ListNode));
    newNode->val = n;
    newNode->next = NULL;

    if (p <= 0) {
        newNode->next = head;
        return newNode;
    }

    struct ListNode *current = head;
    for (int i = 1; i < p - 1 && current; ++i) {
        current = current->next;
    }

    if (!current) {
        printf("Invalid position\n");
        free(newNode);
        return head;
    }

    newNode->next = current->next;
    current->next = newNode;
    return head;
}
```

```
void printList(struct ListNode* head) {
    while (head) {
        printf("%d ", head->val);
        head = head->next;
    }
    printf("\n");
}

int main() {
    struct ListNode *head = malloc(sizeof(struct ListNode));
    head->val = 1;
    head->next = NULL;

    int p = 0, n = 0;

    printf("Original list: ");
    printList(head);

    head = insertNode(head, p, n);

    printf("List after inserting %d at position %d: ", n, p);
    printList(head);

    return 0;
}
```

You are given an undirected graph G(V, E) with N vertices and M edges. We need to
find the minimum number of edges between a given pair of vertices (u, v).
Examples:

Input: For given graph G. Find minimum number of edges between (1, 5).



Output: 2
Explanation: (1, 2) and (2, 5) are the only edges resulting into shortest path between 1 and 5.
You are given an undirected graph G(V, E) with N vertices and M edges. We need to
find the minimum number of edges between a given pair of vertices (u, v).
Examples:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
typedef struct {
    int data[MAX_VERTICES];
    int front, rear;
} Queue;


void initQueue(Queue *q) {
    q->front = q->rear = -1;
}


int isEmpty(Queue *q) {
    return q->front == -1;
}

// Function to enqueue a node to the queue
void enqueue(Queue *q, int value) {
    q->data[++q->rear] = value;
    if (q->front == -1) q->front++;
}

// Function to dequeue a node from the queue
int dequeue(Queue *q) {
    int data = q->data[q->front++];
    if (q->front > q->rear) q->front = q->rear = -1;
    return data;
}

// Function to perform BFS and find the minimum number of edges between source and destination
int BFS(int graph[][MAX_VERTICES], int V, int src, int dest) {
    int visited[MAX_VERTICES] = {0};
    Queue q;
    initQueue(&q);
    enqueue(&q, src);
```

```
    visited[src] = 1;
    int level = 0;
    while (!isEmpty(&q)) {
        int size = q.rear - q.front + 1;
        for (int i = 0; i < size; ++i) {
            int u = dequeue(&q);
            if (u == dest) return level;
            for (int v = 0; v < V; ++v)
                if (graph[u][v] && !visited[v]) enqueue(&q, v), visited[v] = 1;
        }
        level++;
    }
    return -1; // Destination not reachable from source
}

int main() {
    int V = 5; // Number of vertices
    int graph[MAX_VERTICES][MAX_VERTICES] = {
        {0, 1, 0, 0, 0}, // Example: Graph with edges 1-2, 2-3, 2-5, 3-4
        {1, 0, 1, 0, 1},
        {0, 1, 0, 0, 0},
        {0, 0, 0, 0, 1},
        {0, 1, 0, 1, 0}
    };

    int src = 1; // Source vertex
    int dest = 5; // Destination vertex

    int minEdges = BFS(graph, V, src, dest);
    if (minEdges != -1)
        printf("Minimum number of edges between %d and %d: %d\n", src, dest, minEdges);
    else
        printf("No path found between %d and %d\n", src, dest);

    return 0;
}
```

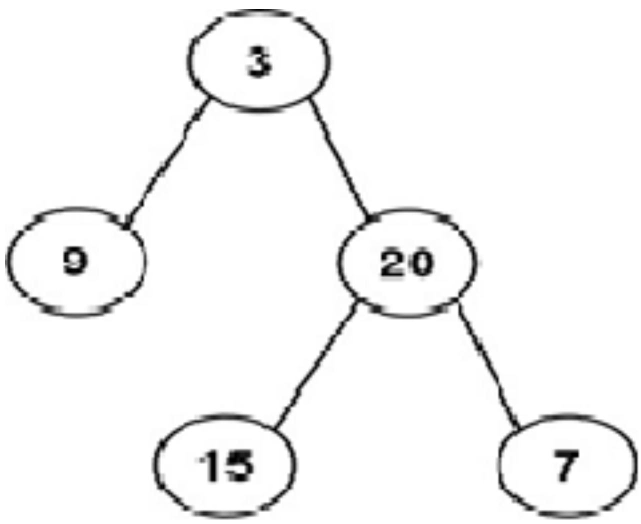
Given the head of a singly linked list, return number of nodes present in a linked

Example 1:
1->2->3->5->8
Output 5

```
#include <stdio.h>
#include <stdlib.h>
struct ListNode
{
    int val;
    struct ListNode *next;
};
int countNodes(struct ListNode* head) {
```

```
    int count = 0;
    while (head != NULL) {
        count++;
        head = head->next;
    }
    return count;
}
int main()
{
    struct ListNode *head = malloc(sizeof(struct ListNode));
    head->val = 1;
    head->next = malloc(sizeof(struct ListNode));
    head->next->val = 2;
    head->next->next = malloc(sizeof(struct ListNode));
    head->next->next->val = 3;
    head->next->next->next = malloc(sizeof(struct ListNode));
    head->next->next->next->val = 5;
    head->next->next->next->next = malloc(sizeof(struct ListNode));
    head->next->next->next->next->val = 8;
    head->next->next->next->next->next = NULL;
    printf("Number of nodes: %d\n", countNodes(head));
    struct ListNode *current = head;
    struct ListNode *temp;
    while (current != NULL)
    {
        temp = current;
        current = current->next;
        free(temp);
    }
  return 0;
}
```

Write a program to traverse the nodes present in the following tree in inorder and postorder traversal



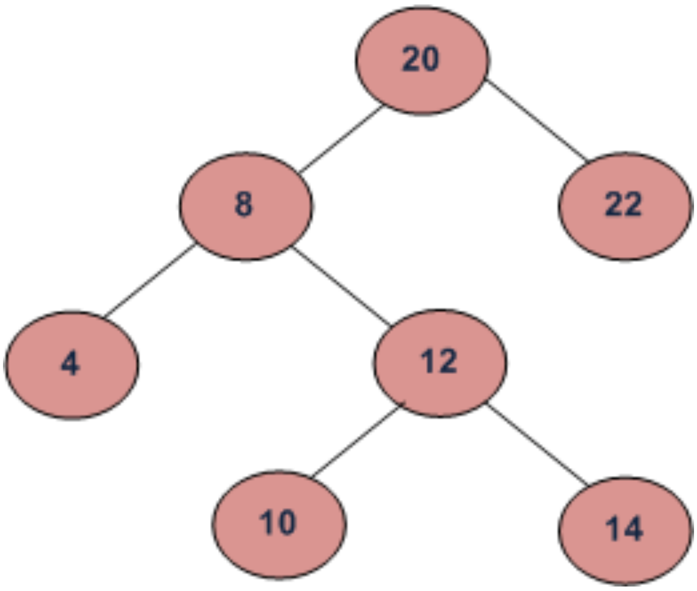

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode
{
```

```
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};

struct TreeNode* newTreeNode(int val)
{
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = node->right = NULL;
    return node;
}

void inorderTraversal(struct TreeNode* root)
{
    if (root)
    {
        inorderTraversal(root->left);
        printf("%d ", root->val);
        inorderTraversal(root->right);
    }
}

void postorderTraversal(struct TreeNode* root) {
    if (root)
    {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->val);
    }
}

int main()
{
    struct TreeNode* root = newTreeNode(3);
    root->left = newTreeNode(9);
    root->right = newTreeNode(20);
    root->left->left = newTreeNode(15);
    root->left->right = newTreeNode(7);

    printf("Inorder: ");
    inorderTraversal(root);
    printf("\n");

    printf("Postorder: ");
    postorderTraversal(root);
    printf("\n");

    return 0;
}
```

Given a string s, sort it in ascending order and find the starting index of repeated character
Input: s = "tree"
Output: "eert", starting index 0
Input: s = "kkj"
Output: "jkk", starting index : 1
Example 2:
Input: s = "cccaaa"
Output: "aaaccc", starting index 0,3
Example 3:
Input: s = "Aabb"
Output: "bbAa",starting index 0,2

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

void findRepeatedCharacter(char *str) {
    int count[256] = {0};
    for (int i = 0; str[i]; i++) count[str[i]]++;
    for (int i = 0; str[i]; i++) {
        if (count[str[i]] > 1) {
            printf("\"");
            for (int j = 0; str[j]; j++) printf("%c", str[j]);
            printf("\", starting index %d\n", i);
            return;
        }
    }
    printf("No repeated character found.\n");
}

int main() {
    char s[100];
    printf("Input: s = ");
    fgets(s, sizeof(s), stdin);
    s[strcspn(s, "\n")] = '\0';
    findRepeatedCharacter(s);
    return 0;
}
```

Given the root of a binary search tree and K as input, find Kth smallest element in BST. For example, in the following BST,



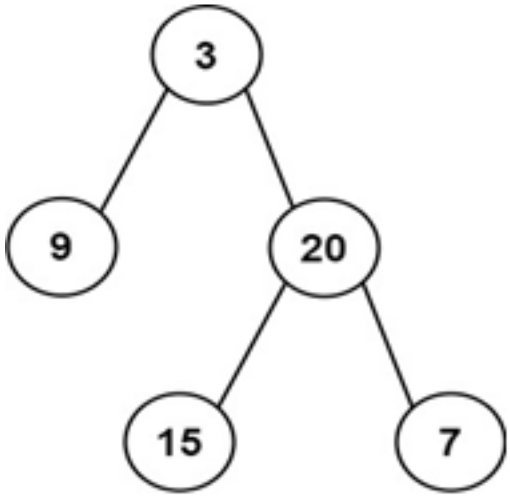

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *left, *right;
};

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

void kthSmallestUtil(struct Node* root, int k, int* count, int* result) {
    if (root == NULL || *count >= k) {
        return;
    }

    kthSmallestUtil(root->left, k, count, result);

    (*count)++;
    if (*count == k) {
        *result = root->data;
        return;
    }

    kthSmallestUtil(root->right, k, count, result);
}

int kthSmallest(struct Node* root, int k) {
    int count = 0;
    int result = -1;
    kthSmallestUtil(root, k, &count, &result);
    return result;
}
```

```
int main() {
    struct Node* root = createNode(20);
    root->left = createNode(8);
    root->right = createNode(22);
    root->left->left = createNode(4);
    root->left->right = createNode(12);
    root->left->right->left = createNode(10);
    root->left->right->right = createNode(14);

    int k = 3;
    printf("Kth smallest element for k = %d is: %d\n", k, kthSmallest(root, k));

    k = 5;
    printf("Kth smallest element for k = %d is: %d\n", k, kthSmallest(root, k));

    return 0;
}
```

Given an unsorted array arr[] with both positive and negative elements, the task is to find the smallest positive number missing from the array.
Input: arr[] = {2, 3, 7, 6, 8, -1, -10, 15}
Output: 1
Input: arr[] = { 2, 3, -7, 6, 8, 1, -10, 15 }
Output: 4
Input: arr[] = {1, 1, 0, -1, -2}
Output: 2

```
#include <stdio.h>
int findMissingPositive(int arr[], int size) {
    int found[size + 1];
    for (int i = 0; i <= size; i++)
        found[i] = 0;
    for (int i = 0; i < size; i++)
        if (arr[i] > 0 && arr[i] <= size)
            found[arr[i]] = 1;
    for (int i = 1; i <= size; i++)
        if (!found[i])
            return i;
    return size + 1;
}

int main() {
    printf("Output for arr1: %d\n", findMissingPositive((int[]){2, 3, 7, 6, 8, -1, -10, 15}, 8));
    printf("Output for arr2: %d\n", findMissingPositive((int[]){2, 3, -7, 6, 8, 1, -10, 15}, 8));
    printf("Output for arr3: %d\n", findMissingPositive((int[]){1, 1, 0, -1, -2}, 5));
    return 0;
}
```

Given two integer arrays preorder and inorder where preorder is the preorder
traversal of a binary tree and inorder is the inorder traversal of the same tree,
construct and return the binary tree.




```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};

struct TreeNode* buildTree(int* preorder, int* inorder, int inStart, int inEnd, int* preIndex) {
    if (inStart > inEnd)
        return NULL;

    struct TreeNode* root = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    root->val = preorder[(*preIndex)++];

    int inIndex;
    for (inIndex = inStart; inIndex <= inEnd; inIndex++) {
        if (inorder[inIndex] == root->val)
            break;
    }

    root->left = buildTree(preorder, inorder, inStart, inIndex - 1, preIndex);
    root->right = buildTree(preorder, inorder, inIndex + 1, inEnd, preIndex);

    return root;
}

void printLevelOrder(struct TreeNode* root) {
    if (root == NULL)
        return;

    struct TreeNode* queue[100];
    int front = -1, rear = -1;
    queue[++rear] = root;
```

```
    while (front != rear) {
        struct TreeNode* temp = queue[++front];

        if (temp != NULL) {
            printf("%d", temp->val);

            if (temp->left || temp->right) {
                queue[++rear] = temp->left;
                queue[++rear] = temp->right;
            }

            if (front != rear)
                printf(",");
        } else {
            printf("null");
            if (front != rear)
                printf(",");
        }
    }
}

int main() {
    int preorder[] = {3, 9, 20, 15, 7};
    int inorder[] = {9, 3, 15, 20, 7};
    int n = sizeof(preorder) / sizeof(preorder[0]);
    int preIndex = 0;

    struct TreeNode* root = buildTree(preorder, inorder, 0, n - 1, &preIndex);

    printf("[");
    printLevelOrder(root);
    printf("]\n");

    return 0;
}
```

Given an array of size N-1 such that it only contains distinct integers in the range of 1 to N. Find the missing element.
Input:
N = 5
A[] = {1,2,3,5}
Output: 4
Input:
N = 10
A[] = {6,1,2,8,3,4,7,10,5}
Output: 9

```
#include <stdio.h>
int findMissingNumber(int arr[], int n)
{
    int xor1 = 0, xor2 = 0;
```

```
    for (int i = 1; i <= n + 1; i++)
        xor1 ^= i;
    for (int i = 0; i < n; i++)
        xor2 ^= arr[i];
    return xor1 ^ xor2;
}

int main()
{
    int N1 = 5;
    int A1[] = {1, 2, 3, 5};
    printf("Missing element for N = %d is %d\n", N1, findMissingNumber(A1, N1 - 1));
    int N2 = 10;
    int A2[] = {6, 1, 2, 8, 3, 4, 7, 10, 5};
    printf("Missing element for N = %d is %d\n", N2, findMissingNumber(A2, N2 - 1));

    return 0;
}
```

Write a program to find odd number present in the data part of a node
Example Linked List 1->2->3->7
Output: 1,3,7

```
#include <stdio.h>
#include <stdlib.h>
struct Node { int data; struct Node* next; };
void findOddNumbers(struct Node* h)
{
    printf("Odd numbers: ");
    while (h) { if (h->data % 2) printf("%d ", h->data); h = h->next; } printf("\n");
}
int main()
{
    struct Node* h = NULL; int v[] = {1, 2, 3, 7};
    for (int i = sizeof(v)/sizeof(v[0])-1; i >= 0; i--) {
        struct Node* n = (struct Node*)malloc(sizeof(struct Node)); n->data = v[i];
        n->next = h; h = n;
    }
    findOddNumbers(h);
    while (h) { struct Node* t = h; h = h->next; free(t); }
    return 0;
}
```

Write a program to perform insert and delete operations in a queue
Example : 12,34,56,78
After insertion of 60 content of the queue is 12,34,56,78,60
After deletion of 12 , the contents of the queue : 34,56,78,60

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

```
#define MAX_SIZE 100

struct Queue {
    int items[MAX_SIZE];
    int front;
    int rear;
};

// Function to initialize the queue
void initQueue(struct Queue *q) {
    q->front = -1;
    q->rear = -1;
}

// Function to check if the queue is full
int isFull(struct Queue *q) {
    return q->rear == MAX_SIZE - 1;
}

// Function to check if the queue is empty
int isEmpty(struct Queue *q) {
    return q->front == -1;
}

// Function to insert an element into the queue
void enqueue(struct Queue *q, int value) {
    if (isFull(q)) {
        printf("Queue is full\n");
    } else {
        if (isEmpty(q)) {
            q->front = 0;
        }
        q->rear++;
        q->items[q->rear] = value;
        printf("Inserted %d\n", value);
    }
}

// Function to remove an element from the queue
void dequeue(struct Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
    } else {
        printf("Deleted %d\n", q->items[q->front]);
        q->front++;
        if (q->front > q->rear) {
            q->front = q->rear = -1;
        }
    }
}

// Function to display the contents of the queue
void display(struct Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
    } else {
        printf("Contents of the queue: ");
        for (int i = q->front; i <= q->rear; i++) {
            printf("%d ", q->items[i]);
        }
```

```
        printf("\n");
    }
}

int main() {
    struct Queue q;
    initQueue(&q);

    enqueue(&q, 12);
    enqueue(&q, 34);
    enqueue(&q, 56);
    enqueue(&q, 78);

    display(&q);

    enqueue(&q, 60);

    display(&q);

    dequeue(&q);

    display(&q);

    return 0;
}
```

Given a string s containing just the characters '(', ')', '{', '}', '['
and ']', determine if the input string is valid.
An input string is valid if:
1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
Input: s = "()"
Output: true
Input: s = "()[]{}"
Output: true
Input: s = "(]"
Output: false
Input: s = "([)]"
Output: false
Input: s = "{[]}"
Output: true

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
bool isValid(char *s) {
    char stack[1000];
    int top = -1;
    for (int i = 0; s[i] != '\0';i++) {
```

```
        if (s[i] == '(' || s[i] == '{' || s[i] == '[') {
            stack[++top] = s[i];
        } else {
            if (top == -1) return false;
            if (s[i] == ')' && stack[top] != '(') return false;
            if (s[i] == '}' && stack[top] != '{') return false;
            if (s[i] == ']' && stack[top] != '[') return false;
            top--;
        }
    }

    return top == -1;
}
int main() {
    char s[1000];
    //printf("Enter a string: ");
    scanf("%s", s);
    if (isValid(s)) {
        printf("True\n");
    } else {
        printf("False\n");
    }
    return 0;
}
```

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.
Implement the MinStack class:
1. MinStack() initializes the stack object.
2. void push(int val) pushes the element val onto the stack.
3. void pop() removes the element on the top of the stack.
4. int top() gets the top element of the stack.
5. int getMin() retrieves the minimum element in the stack.
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
Output
[null,null,null,null,-3,null,0,-2]
Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top();   // return 0
minStack.getMin(); // return -2

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
    int *stack;
    int *minStack;
    int top;
} MinStack;

MinStack* minStackCreate() {
    MinStack* stack = (MinStack*)malloc(sizeof(MinStack));
    stack->stack = (int*)malloc(sizeof(int) * 10000);
    stack->minStack = (int*)malloc(sizeof(int) * 10000);
    stack->top = -1;
    return stack;
}

void minStackPush(MinStack* obj, int val) {
    obj->stack[++obj->top] = val;
    if (obj->top == 0 || val <= obj->minStack[obj->top - 1])
        obj->minStack[obj->top] = val;
    else
        obj->minStack[obj->top] = obj->minStack[obj->top - 1];
}
```

```
void minStackPop(MinStack* obj) {
    obj->top--;
}

int minStackTop(MinStack* obj) {
    return obj->stack[obj->top];
}

int minStackGetMin(MinStack* obj) {
    return obj->minStack[obj->top];
}

void minStackFree(MinStack* obj) {
    free(obj->stack);
    free(obj->minStack);
    free(obj);
}

int main() {
    MinStack* obj = minStackCreate();
    minStackPush(obj, -2);
    minStackPush(obj, 0);
    minStackPush(obj, -3);
    printf("%d\n", minStackGetMin(obj));
    minStackPop(obj);
    printf("%d\n", minStackTop(obj));
    printf("%d\n", minStackGetMin(obj));
    minStackFree(obj);
    return 0;
}
```

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the sum of these two arrays
Example 1:
Input: nums1 = [1,3], nums2 = [2]
Output: 6
Example 2:
Input: nums1 = [1,2], nums2 = [3,4]
Output: 10

```
#include <stdio.h>

int sumOfSortedArrays(int nums1[], int m, int nums2[], int n) {
    int sum = 0, i = 0, j = 0;
    while (i < m || j < n) {
        if (j >= n || (i < m && nums1[i] < nums2[j])) sum += nums1[i++];
        else sum += nums2[j++];
    }
    return sum;
}

int main() {
    int nums1[] = {1, 3}, m = 2;
    int nums2[] = {2}, n = 1;
    printf("Output for Example 1: %d\n", sumOfSortedArrays(nums1, m, nums2, n));
```

```
    int nums3[] = {1, 2}, nums4[] = {3, 4}, m2 = 2, n2 = 2;
    printf("Output for Example 2: %d\n", sumOfSortedArrays(nums3, m2, nums4, n2));

    return 0;
}
```