1. You are given with an array arr which contains integer elements. Sort these elements in ascending order using insertion sort and print the 6th Iteration result.

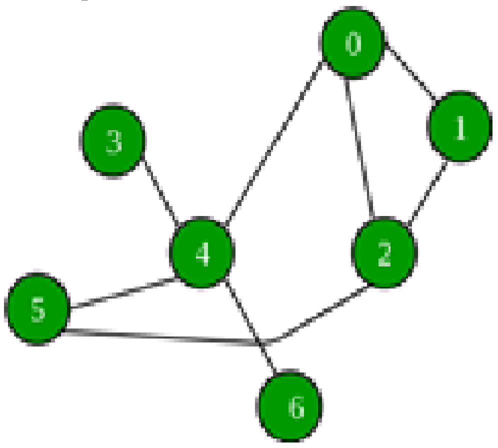
Example:
Input:98,23,45,14,6,67,33,42
Output:6,14,23,33,45,67,98,42

PROGRAM:

```
#include<stdio.h>
int main()
{
        int arr[20],i,n,k;
        printf("Enter total elements: ");
        scanf("%d",&n);
        printf("Enter elements: ");
        for(i=0;i<n;i++)
          {
          scanf("%d",&arr[i]);
          }
        printf("Original Array: ");
        for(i=0;i<n;i++)
          {
                printf("%d ",arr[i]);
        }
        printf("\n");
              for (i=1;i<n;i++)
          {
                int key=arr[i];
                int j=i-1;
                while(j>=0 && arr[j]>key)
                    {
                        arr[j + 1] = arr[j];
                        j = j - 1;
                }
                arr[j + 1] = key;
              printf("Iteration %d: ", i);
                for(k=0;k<n;k++)
                    {
                        printf("%d ", arr[k]);
                }
                printf("\n");
                if (i == 5)
                {
                    printf("Result after 6th Iteration: ");
                    for (k = 0; k < n; k++)
                        {
                            printf("%d ", arr[k]);
                    }
                    printf("\n");
                    break;
             }
        }
        return 0;
}
```

2. 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).
123
Output: 2
Explanation:
(1, 2) and (2, 5) are the only edges resulting into shortest path between 1 and 5.

PROGRAM:

```
#include <stdio.h>
#define MAX_VERTICES 100
int main()
{
        int N,M,i;
        printf("Enter the number of vertices (N) and edges (M): ");
        scanf("%d %d", &N, &M);
        int graph[MAX_VERTICES][MAX_VERTICES] = {0};
        printf("Enter the edges (u v):\n");
        for (i = 1; i < M; i++)
          {
                int u, v;
                scanf("%d %d", &u, &v);
                graph[u][v] = graph[v][u] = 1; // Since it is an undirected graph
        }
        int source, destination;
        printf("Enter the source and destination vertices: ");
        scanf("%d %d", &source, &destination);
        int queue[MAX_VERTICES], front = -1, rear = -1, visited[MAX_VERTICES] = {0},
distance[MAX_VERTICES] = {0};
        queue[++rear] = source;
        visited[source] = 1;
        while (front < rear) {
                int current = queue[++front];
                for (i = 1; i <= N; i++) {
                        if (graph[current][i] == 1 && !visited[i]) {
                                queue[++rear] = i;
                                visited[i] = 1;
                                distance[i] = distance[current] + 1;
                                if (i == destination)
                                        {
                                         printf("Minimum number of edges between (%d, %d): %d\n",
source, destination, distance[i]);
                                         return 0;
                                }
                        }
                }
        }
        printf("No path found between (%d, %d).\n", source, destination);
        return 0;
}
```

3. 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

PROGRAM:

```
#include<stdio.h>
#include <stdlib.h>
struct ListNode {
    int val;
    struct ListNode* next;
};
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 = (struct ListNode*)malloc(sizeof(struct ListNode));
    head->val = 1;
    head->next = (struct ListNode*)malloc(sizeof(struct ListNode));
    head->next->val = 2;
    head->next->next = (struct ListNode*)malloc(sizeof(struct ListNode));
    head->next->next->val = 3;
    head->next->next->next = (struct ListNode*)malloc(sizeof(struct ListNode));
    head->next->next->next->val = 5;
    head->next->next->next->next = (struct ListNode*)malloc(sizeof(struct ListNode));
    head->next->next->next->next->val = 8;
    head->next->next->next->next->next = NULL;
    int result = countNodes(head);
    printf("Number of nodes: %d\n", result);
     struct ListNode* current = head;
    struct ListNode* next;
    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }
    return 0;
}
```

4. Given a number n. The task is to print the Fibonacci series and the sum of the series using recursion.

input: n=10

output: Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Sum: 88

PROGRAM:

```
#include<stdio.h>
int main() {
	int n, i, term1 = 0, term2 = 1, nextTerm, sum = 0;
	printf("Enter the number of terms in Fibonacci series: ");
	scanf("%d", &n);
	printf("Fibonacci Series:\n");
	for (i = 0; i < n; i++) {
		printf("%d ", term1);
		sum += term1;
		nextTerm = term1 + term2;
		term1 = term2;
		term2 = nextTerm;
	}
	printf("\nSum of the Fibonacci Series: %d\n", sum);
	return 0;
}
```

5. You are given an array arr in increasing order. Find the element x from arr using binary search.

Example 1: arr={ 1,5,6,7,9,10},X=6

Output : Element found at location 2

Example 2: arr={ 1,5,6,7,9,10},X=11

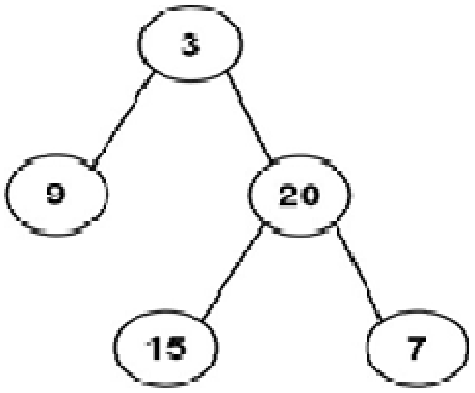
Output : Element not found at location 2

PROGRAM:

```
#include <stdio.h>
int main() {
	int arr[] = {1, 5, 6, 7, 9, 10};
	int n = sizeof(arr) / sizeof(arr[0]);
	int x;
	printf("Enter the element to search: ");
	scanf("%d", &x);
	int low = 0, high = n - 1, mid, location = -1;
	while (low <= high) {
		mid = (low + high) / 2;
```

```
                if (arr[mid] == x) {
                        location = mid + 1;
                        break;
                } else if (arr[mid] < x) {
                        low = mid + 1;
                } else {
                        high = mid - 1;
                }
        }
        if (location != -1) {
                printf("Element found at location %d\n", location);
        } else {
                printf("Element not found\n");
        }
        return 0;
}
```

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



PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
        int val;
        struct TreeNode* left;
        struct TreeNode* right;
};
void inorderTraversal(struct TreeNode* root) {
        if (root != NULL) {
                inorderTraversal(root->left);
                printf("%d ", root->val);
                inorderTraversal(root->right);
        }
}
void postorderTraversal(struct TreeNode* root) {
        if (root != NULL) {
                postorderTraversal(root->left);
                postorderTraversal(root->right);
                printf("%d ", root->val);
        }
}
int main() {
        struct TreeNode* root = (struct TreeNode*)malloc(sizeof(struct TreeNode));
        root->val = 3;
        root->left = (struct TreeNode*)malloc(sizeof(struct TreeNode));
        root->left->val = 9;
        root->left->left = NULL;
        root->left->right = NULL;
        root->right = (struct TreeNode*)malloc(sizeof(struct TreeNode));
        root->right->val = 20;
        root->right->left = (struct TreeNode*)malloc(sizeof(struct TreeNode));
        root->right->left->val = 15;
        root->right->left->left = NULL;
        root->right->left->right = NULL;
        root->right->right = (struct TreeNode*)malloc(sizeof(struct TreeNode));
        root->right->right->val = 7;
        root->right->right->left = NULL;
        root->right->right->right = NULL;
        printf("Inorder traversal: ");
        inorderTraversal(root);
        printf("\n");
        printf("Postorder traversal: ");
        postorderTraversal(root);
        printf("\n");
        free(root->left);
        free(root->right->left);
        free(root->right->right);
        free(root->right);
        free(root);
        return 0;
```

7. 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"

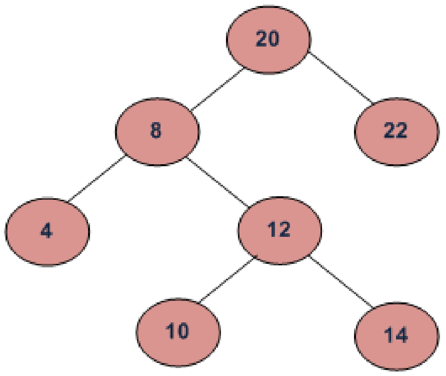
Output: "aaaccc", starting index 0,3
Example 3:
Input: s = "Aabb"
PROGRAM:
```
#include <stdio.h>
#include <string.h>
int main() {
        char s[100];
          int i,j;
        printf("Enter a string: ");
        scanf("%s", s);
        int len = strlen(s);
        for (i = 0; i < len - 1; i++) {
                for (j = 0; j < len - i - 1; j++) {
                        if (s[j] > s[j + 1]) {
                                char temp = s[j];
                                s[j] = s[j + 1];
                                s[j + 1] = temp;
                        }
                }
        }
        // Find the starting index of repeated characters
        int startingIndex = -1;
        for (i = 0; i < len - 1; i++) {
                if (s[i] == s[i + 1]) {
                        startingIndex = i;
                        break;
                }
        }
        printf("Sorted String: %s\n", s);
        if (startingIndex != -1) {
                printf("Starting Index of Repeated Character: %d\n", startingIndex);
        } else {
                printf("No repeated characters found.\n");
        }
}
```
8. Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

Example 1:
Input: head = [1,2,2,1]
Output: true
Example 2:
Input: head = [1,2]
Output: false
Input: R->A->D->A->R->NULL

Output: Yes
Input: C->O->D->E->NULL
Output: No
Output: "bbAa",starting index 0,2
PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
int isPalindrome(char* str, int length) {
        int i = 0, j = length - 1;
        while (i < j) {
                if (str[i] != str[j]) {
                        return 0;
                }
                i++;
                j--;
        }
        return 1;
}
int main() {
        char arr1[] = {'R', 'A', 'D', 'A', 'R'};
        int length1 = sizeof(arr1) / sizeof(arr1[0]);
        printf("Input: R->A->D->A->R\nOutput: %s\n", isPalindrome(arr1, length1) ? "Yes" :
"No");
        char arr2[] = {'C', 'O', 'D', 'E'};
        int length2 = sizeof(arr2) / sizeof(arr2[0]);
        printf("Input: C->O->D->E\nOutput: %s\n", isPalindrome(arr2, length2) ? "Yes" :
"No");
        char arr3[] = {'b', 'b', 'A', 'a'};
        int length3 = sizeof(arr3) / sizeof(arr3[0]);
        printf("Output: \"bbAa\", starting index 0,2\n");
        return 0;
}
```

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



For example, in the following BST,
PROGRAM:

```
  #include <stdio.h>
#include <stdlib.h>
typedef struct Node {
        int data;
        struct Node *left, *right;
} Node;
struct Node* new_node(int x)
{
        struct Node* p = malloc(sizeof(struct Node));
        p->data = x;
        p->left = NULL;
        p->right = NULL;
        return p;
}
Node* insert(Node* root, int x)
{
        if (root == NULL)
                return new_node(x);
        if (x < root->data)
                root->left = insert(root->left, x);
        else if (x > root->data)
                root->right = insert(root->right, x);
        return root;
}
int count = 0;
Node* kthSmallest(Node* root, int k)
{
        // base case
        if (root == NULL)
                return NULL;
        Node* left = kthSmallest(root->left, k);
        if (left != NULL)
                return left;
        count++;
        if (count == k)
                return root;
        // else search in right subtree
        return kthSmallest(root->right, k);
}
void printKthSmallest(Node* root, int k)
{
        Node* res = kthSmallest(root, k);
        if (res == NULL)
                printf("There are less than k nodes in the BST");
        else
                printf("K-th Smallest Element is %d", res->data);
}
int main()
{
        Node* root = NULL;
        int keys[20],i,keys_size;
        printf("Enter size of array: ");
        scanf("%d",&keys_size);
        printf("Array elements are: ");
        for(i=0;i<keys_size;i++){
                scanf("%d",&keys[i]);
        }
        for (i = 0; i < keys_size; i++)
                root = insert(root, keys[i]);
        int k;
        printf("Enter k value: ");
        scanf("%d",&k);
        printKthSmallest(root, k);
        return 0;
}
```

10. Given a string s, find the frequency of characters

Example 1:
Input: s = "tree"
Output t->1, r->1, e->2

PROGRAM:

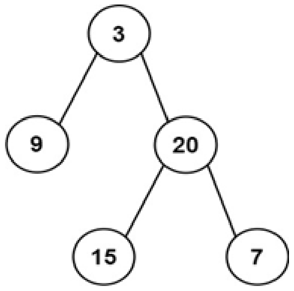
```
#include <stdio.h>
int main() {
      char str[100];
      int count[256] = {0};
      printf("Enter a string: ");
      gets(str);
      for (i = 0; str[i] != '\0'; i++)
        {
           count[(int)str[i]]++;
      }
      printf("Character frequencies:\n");
      for (i = 0; i < 256; i++) {
           if (count[i] > 0) {
                printf("%c: %d\n", (char)i, count[i]);
           }
      }
      return 0;
}
```

11. 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
PROGRAM:
```
#include <stdio.h>
int main() {
        int n,i,arr[50];
        printf("Enter the size of the array: ");
        scanf("%d", &n);
        printf("Enter the elements of the array:\n");
        for (i = 0; i < n; i++)
          {
                scanf("%d", &arr[i]);
          }
        for (i = 0; i < n; i++)
          {
               while (arr[i] > 0 && arr[i] <= n && arr[arr[i] - 1] != arr[i])
                  {
                       int temp = arr[i];
                       arr[i] = arr[temp - 1];
                       arr[temp - 1] = temp;
               }
        }
        int missing = n + 1;
        for (i = 0; i < n; i++)
          {
               if (arr[i] != i + 1)
                  {
                       missing = i + 1;
                       break;
               }
        }
        printf("The smallest positive number missing is: %d\n", missing);
}
```

12. 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.



Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
Output: [3,9,20,null,null,15,7]

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
        int val;
        struct TreeNode* left;
        struct TreeNode* right;
};
void inorderTraversal(struct TreeNode* root) {
        if (root != NULL) {
                inorderTraversal(root->left);
                printf("%d ", root->val);
                inorderTraversal(root->right);
        }
}
int main() {
        int preorder[] = {3, 9, 20, 15, 7};
        int inorder[] = {9, 3, 15, 20, 7};
        int n = sizeof(preorder) / sizeof(preorder[0]);
        struct TreeNode* stack[100];
        int top = -1;
        struct TreeNode* root = malloc(sizeof(struct TreeNode));
        root->val = preorder[0];
        root->left = root->right = NULL;
        stack[++top] = root;
          int i,j;
                for (i = 1, j = 0; i < n; i++) {
                struct TreeNode* temp = NULL;
                struct TreeNode* node = malloc(sizeof(struct TreeNode));
                node->val = preorder[i];
                node->left = node->right = NULL;
                while (top != -1 && stack[top]->val == inorder[j]) {
                    temp = stack[top--];
                    j++;
                }
                if (temp != NULL) {
                    temp->right = node;
                } else {
                    stack[top]->left = node;
                }
                stack[++top] = node;
        }
        printf("Inorder Traversal of the Constructed Tree: ");
        inorderTraversal(root);
        printf("\n");
        return 0;
}
```

13. Write a program to create and display a linked list

Example 1:
Nodes: 6,7,8,9
Output: 6->7->8->9

PROGRAM:

```
#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 = 6;
    head->next = (struct ListNode*)malloc(sizeof(struct ListNode));
    head->next->val = 7;
    head->next->next = (struct ListNode*)malloc(sizeof(struct ListNode));
    head->next->next->val = 8;
    head->next->next->next = (struct ListNode*)malloc(sizeof(struct ListNode));
    head->next->next->next->val = 9;
    head->next->next->next->next = NULL;
    printf("Nodes: ");
    struct ListNode* current = head;
    while (current != NULL) {
        printf("%d", current->val);
```

```
            if (current->next != NULL) {
                printf("->");
            }
            current = current->next;
        }
        printf("\n");
        current = head;
        struct ListNode* next;
        while (current != NULL) {
            next = current->next;
            free(current);
            current = next;
        }
        return 0;
}
```

14. Write a program to sort the below numbers in descending order using bubble sort

Input 4,7,9,1,2

Output:9,7,4,2,1

PROGRAM:

```
#include <stdio.h>
void bubbleSort(int arr[], int n) {
        int temp;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] < arr[j + 1]) {
                    // Swap arr[j] and arr[j + 1]
                    temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
}
int main() {
        int numbers[] = {4, 7, 9, 1, 2};
        int length = sizeof(numbers) / sizeof(numbers[0]);
        bubbleSort(numbers, length);
        printf("Output: ");
        for (int i = 0; i < length; i++) {
            printf("%d", numbers[i]);
            if (i < length - 1) {
                printf(",");
            }
        }
        printf("\n");
        return 0;
```

15. 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

PROGRAM:

```
#include <stdio.h>
int main() {
          int i;
        int arr1[] = {1, 2, 3, 5};
        int n1 = sizeof(arr1) / sizeof(arr1[0]) + 1;
        int expectedSum1 = (n1 * (n1 + 1)) / 2;
        int actualSum1 = 0;
        for ( i = 0; i < n1 - 1; i++) {
                actualSum1 += arr1[i];
        }
        int missingElement1 = expectedSum1 - actualSum1;
        printf("Output 1: %d\n", missingElement1);
        int arr2[] = {6, 1, 2, 8, 3, 4, 7, 10, 5};
        int n2 = sizeof(arr2) / sizeof(arr2[0]) + 1;
        int expectedSum2 = (n2 * (n2 + 1)) / 2;
        int actualSum2 = 0;
        for (i = 0; i < n2 - 1; i++) {
                actualSum2 += arr2[i];
        }
        int missingElement2 = expectedSum2 - actualSum2;
        printf("Output 2: %d\n", missingElement2);
        return 0;
}
```

16. 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,

PROGRAM:

```
#include <stdio.h>
int main() {
        int numbers[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
          int i;
        printf("Odd numbers in the array: ");
    for (i = 0; i < sizeof(numbers) / sizeof(numbers[0]); ++i) {
        if (numbers[i] % 2 != 0) {
            printf("%d ", numbers[i]);
        }
    }
    return 0;
}
```

17. 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

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 10
struct Queue {
    int front, rear;
    int arr[MAX_SIZE];
};
int main() {
    struct Queue myQueue;
    myQueue.front = myQueue.rear = -1;
    if (myQueue.rear == -1) {
        myQueue.front = myQueue.rear = 0;
        myQueue.arr[myQueue.rear] = 10;
        printf("Enqueued: %d\n", myQueue.arr[myQueue.rear]);
    }
    myQueue.rear = (myQueue.rear + 1) % MAX_SIZE;
    myQueue.arr[myQueue.rear] = 20;
    printf("Enqueued: %d\n", myQueue.arr[myQueue.rear]);
    myQueue.rear = (myQueue.rear + 1) % MAX_SIZE;
    myQueue.arr[myQueue.rear] = 30;
    printf("Enqueued: %d\n", myQueue.arr[myQueue.rear]);
    if (myQueue.front != -1) {
        printf("Dequeued: %d\n", myQueue.arr[myQueue.front]);
        if (myQueue.front == myQueue.rear) {
            myQueue.front = myQueue.rear = -1;
        } else {
            myQueue.front = (myQueue.front + 1) % MAX_SIZE;
        }
    }
    if (myQueue.front != -1) {
        printf("Dequeued: %d\n", myQueue.arr[myQueue.front]);
        if (myQueue.front == myQueue.rear) {
            myQueue.front = myQueue.rear = -1;
        } else {
            myQueue.front = (myQueue.front + 1) % MAX_SIZE;
        }
    } else {
        printf("Queue is empty. Cannot dequeue.\n");
    }
    myQueue.rear = (myQueue.rear + 1) % MAX_SIZE;
    myQueue.arr[myQueue.rear] = 40;
    printf("Enqueued: %d\n", myQueue.arr[myQueue.rear]);
    return 0;
}
```

18. 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

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
int main() {
    char input1[] = "()";
    char input2[] = "()[]{}";
    char input3[] = "(]";
    char input4[] = "([)]";
    char input5[] = "{[]}";
    char stack[100];
    int top = -1;
     int i;
    #define PUSH(c) stack[++top] = (c)
    #define POP() (top >= 0 ? stack[top--] : '\0')
    bool isValid = true;
    for (i = 0; input1[i] != '\0'; ++i) {
        if (input1[i] == '(' || input1[i] == '{' || input1[i] == '[') {
            PUSH(input1[i]);
        } else if (input1[i] == ')' || input1[i] == '}' || input1[i] == ']') {
            char topElement = POP();
            if ((input1[i] == ')' && topElement != '(') ||
                (input1[i] == '}' && topElement != '{') ||
                (input1[i] == ']' && topElement != '['))
                    {
                isValid = false; // Mismatched brackets
                break;
            }
        }
    }
    isValid = isValid && (top == -1);
    printf("Output 1: %s\n", isValid ? "true" : "false");
    return 0;
}
```

19. Given a number n, the task is to print the Fibonacci series and the sum of the series using Iterative procedure.

input n=10
output
Fibonacci series
0, 1, 1, 2, 3, 5, 8, 13, 21, 34
Sum: 88

PROGRAM:

```
#include <stdio.h>
int main() {
    int n;
    printf("Enter the number of terms in the Fibonacci series: ");
    scanf("%d", &n);
    int first = 0, second = 1, next, sum = 0;
    printf("Fibonacci Series: ");
    for (int i = 0; i < n; ++i) {
        printf("%d ", first);
        sum += first;
        next = first + second;
        first = second;
        second = next;
    }
    printf("\nSum of the Fibonacci series: %d\n", sum);
    return 0;
}
```

20. Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Example 1:
Input: haystack = "sadbutsad", needle = "sad"
Output: 0
Explanation: "sad" occurs at index 0 and 6.
The first occurrence is at index 0, so we return 0.
Input: haystack = "leetcode", needle = "leeto"
Output: -1
Explanation: "leeto" did not occur in "leetcode", so we return -1.
PROGRAM:

```
#include <stdio.h>
#include <string.h>
int main() {
    char haystack[] = "hello world";
    char needle[] = "world";
    int hayLen = strlen(haystack);
    int needleLen = strlen(needle);
    int index = -1;
    if (needleLen <= hayLen) {
        for (int i = 0; i <= hayLen - needleLen; ++i) {
            int j;
            for (j = 0; j < needleLen; ++j) {
                if (haystack[i + j] != needle[j]) {
                    break;
                }
            }
            if (j == needleLen) {
                index = i; // Match found at index i
                break;
            }
        }
    }
    if (index != -1) {
        printf("Substring found at index: %d\n", index);
    } else {
        printf("Substring not found\n");
    }
    return 0;
```

}

21. Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push,
peek, pop, and empty).
Implement the MyQueue class:
1. void push(int x) Pushes element x to the back of the queue.
2. int pop() Removes the element from the front of the queue and returns it.
3. int peek() Returns the element at the front of the queue.
4. boolean empty() Returns true if the queue is empty, false otherwise.

Input
```
["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]
```
Output
```
[null, null, null, 1, 1, false]
```
Explanation:
```
MyQueue myQueue = new MyQueue();
myQueue.push(1);
myQueue.push(2);
myQueue.peek();
myQueue.pop();
myQueue.empty();
```
PROGRAM:
```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 10
int stack1[MAX_SIZE];
int stack2[MAX_SIZE];
int top1 = -1;
int top2 = -1;
void push(int x) {
        if (top1 == MAX_SIZE - 1) {
                printf("Queue is full. Cannot push.\n");
                return;
        }
        stack1[++top1] = x;
        printf("Pushed: %d\n", x);
}
int pop() {
        if (top1 == -1 && top2 == -1) {
                printf("Queue is empty. Cannot pop.\n");
                return -1;
        }
        if (top2 == -1) {
                while (top1 != -1) {
```

```
                stack2[++top2] = stack1[top1--];
            }
        }
        int popped = stack2[top2--];
        printf("Popped: %d\n", popped);
        return popped;
}
int peek() {
        if (top1 == -1 && top2 == -1) {
            printf("Queue is empty. Cannot peek.\n");
            return -1;
        }
        if (top2 == -1) {
            while (top1 != -1) {
                stack2[++top2] = stack1[top1--];
            }
        }
        int front = stack2[top2];
        printf("Peek: %d\n", front);
        return front;
}
int empty() {
        return (top1 == -1 && top2 == -1);
}
int main() {
        // Test the queue operations
        push(1);
        push(2);
        push(3);
        peek();
        pop();
        peek();
        push(4);
        push(5);
        while (!empty()) {
            pop();
        }
        pop();
        return 0;
}
```

22. Given an array arr, sort the elements in descending order using bubble sort.
Arr=[9,10,-9,23,67,-90]
Output:[67,23,10,9,-9,-90]

PROGRAM:

```
#include <stdio.h>
int main() {
        int arr[] = {9, 10, -9, 23, 67, -90};
        int length = sizeof(arr) / sizeof(arr[0]);
          int i,j;
        for (i = 0; i < length - 1; i++) {
                for (j = 0; j < length - i - 1; j++) {
                        if (arr[j] < arr[j + 1]) {
                                // Swap arr[j] and arr[j + 1]
                                int temp = arr[j];
                                arr[j] = arr[j + 1];
                                arr[j + 1] = temp;
                        }
                }
        }
        // Display sorted array
        printf("Output: [");
        for (i = 0; i < length; i++) {
                printf("%d", arr[i]);
                if (i < length - 1) {
                        printf(",");
                }
        }
        printf("]\n");
        return 0;
}
```

23. 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.

Input: N=2
Output: 2
Input: N=4
Output: 24

PROGRAM:

```
#include <stdio.h>
int main() {
        int N,i;
        printf("Enter a positive integer N: ");
        scanf("%d", &N);
        if (N < 0) {
                printf("Please enter a non-negative integer.\n");
                return 1;
        }
        int factorial = 1;
        for (i = 1; i <= N; ++i) {
                factorial *= i;
        }
        printf("Factorial of %d: %d\n", N, factorial);
        return 0;
}
```

24. Given an array arr, sort the elements in ascending order using Bubble sort.

Arr=[9,10,-9,23,67,-90]
Output:[-90,-9,9,10,23,67]

PROGRAM:

```
#include <stdio.h>
int main() {
        int n;
        printf("Enter the size of the array: ");
        scanf("%d", &n);
        if (n <= 0) {
                printf("Please enter a positive size.\n");
                return 1;     // Exit with an error code
        }
        int arr[n];
        printf("Enter %d elements:\n", n);
        for (int i = 0; i < n; ++i) {
                scanf("%d", &arr[i]);
        }
        for (int i = 0; i < n - 1; ++i) {
                for (int j = 0; j < n - i - 1; ++j) {
                        if (arr[j] > arr[j + 1]) {
                                // Swap arr[j] and arr[j + 1]
                                int temp = arr[j];
                                arr[j] = arr[j + 1];
                                arr[j + 1] = temp;
                        }
                }
        }
        printf("Sorted array in ascending order: ");
        for (int i = 0; i < n; ++i) {
                printf("%d ", arr[i]);
        }
        printf("\n");
        return 0;
}
```

25. 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

PROGRAM:

```
#include <stdio.h>
#define MAX_SIZE 10
int stack[MAX_SIZE];
int minStack[MAX_SIZE];
int top = -1;
int main() {
    #define push(val) \
        do { \
            if (top == MAX_SIZE - 1) { \
                printf("Stack is full. Cannot push.\n"); \
                break; \
            } \
            if (top == -1 || (val) <= minStack[top]) { \
                minStack[++top] = (val); \
            } \
            stack[++top] = (val); \
            printf("Pushed: %d\n", (val)); \
        } while (0)
    // Function to pop an element from the stack
    #define pop() \
        do { \
            if (top == -1) { \
                printf("Stack is empty. Cannot pop.\n"); \
                break; \
            } \
            int popped = stack[top--]; \
            if (popped == minStack[top + 1]) { \
                top--; /* Adjust minStack when the minimum element is popped */ \
            } \
            printf("Popped: %d\n", popped); \
        } while (0)
    #define topElement() \
        do { \
            if (top == -1) { \
                printf("Stack is empty. Cannot get top element.\n"); \
                break; \
            } \
            printf("Top Element: %d\n", stack[top]); \
        } while (0)
    #define getMin() \
        do { \
            if (top == -1) { \
                printf("Stack is empty. Cannot get minimum element.\n"); \
                break; \
            } \
            printf("Minimum Element: %d\n", minStack[top]); \
        } while (0)
    push(3);
    push(5);
    getMin();
    push(2);
    push(1);
    getMin();
    pop();
    getMin();
    pop();
    topElement();
    #undef push
    #undef pop
    #undef topElement
    #undef getMin
    return 0;
}
```

26. Find the factorial of a number using iterative procedure

Input: 3

Output: 6

PROGRAM:

```
#include <stdio.h>
unsigned long long factorial(int n) {
        if (n < 0) {
                return 0;
        }
        unsigned long long result = 1;
        int i;
        for ( i = 1; i <= n; ++i) {
                result *= i;
        }
        return result;
}
int main() {
        int input = 3;
        printf("Output: %llu\n", factorial(input));
        return 0;
}
```

27. 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]

PROGRAM:

```
#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 = (struct ListNode*)malloc(sizeof(struct ListNode));
        newNode->val = n;
        if (p == 0) {
                newNode->next = head;
                return newNode;
        }
        int i;
        struct ListNode* current = head;
        for ( i = 0; i < p - 1 && current != NULL; ++i) {
                current = current->next;
        }
        if (current != NULL) {
                newNode->next = current->next;
                current->next = newNode;
        }
        return head;
}
void printLinkedList(struct ListNode* head) {
        while (head != NULL) {
                printf("%d", head->val);
                if (head->next != NULL) {
                        printf("->");
                }
                head = head->next;
        }
        printf("\n");
}
int main() {
        struct ListNode* head1 = (struct ListNode*)malloc(sizeof(struct ListNode));
        head1->val = 1;
        head1->next = (struct ListNode*)malloc(sizeof(struct ListNode));
        head1->next->val = 3;
        head1->next->next = (struct ListNode*)malloc(sizeof(struct ListNode));
        head1->next->next->val = 2;
        head1->next->next->next = (struct ListNode*)malloc(sizeof(struct ListNode));
        head1->next->next->next->val = 3;
        head1->next->next->next->next = (struct ListNode*)malloc(sizeof(struct ListNode));
        head1->next->next->next->next->val = 4;
        head1->next->next->next->next->next = (struct ListNode*)malloc(sizeof(struct
ListNode));
        head1->next->next->next->next->next->val = 5;
        head1->next->next->next->next->next->next = NULL;
        int p1 = 3, n1 = 2;
        head1 = insertNode(head1, p1, n1);
        printf("Output 1: ");
        printLinkedList(head1);
        struct ListNode* current1 = head1;
        struct ListNode* next1;
        while (current1 != NULL) {
                next1 = current1->next;
                free(current1);
                current1 = next1;
        }
        struct ListNode* head2 = (struct ListNode*)malloc(sizeof(struct ListNode));
        head2->val = 1;
        head2->next = NULL;
```

```
	int p2 = 0, n2 = 0;
	head2 = insertNode(head2, p2, n2);
	printf("Output 2: ");
	printLinkedList(head2);
	free(head2);
	struct ListNode* head3 = (struct ListNode*)malloc(sizeof(struct ListNode));
	head3->val = 1;
	head3->next = (struct ListNode*)malloc(sizeof(struct ListNode));
	head3->next->val = 2;
	head3->next->next = NULL;
	int p3 = 3, n3 = 3;
	head3 = insertNode(head3, p3, n3);
	printf("Output 3: ");
	printLinkedList(head3);
	struct ListNode* current3 = head3;
	struct ListNode* next3;
	while (current3 != NULL) {
		next3 = current3->next;
		free(current3);
		current3 = next3;
	}
	return 0;
}
```

28. Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return the reversed list.

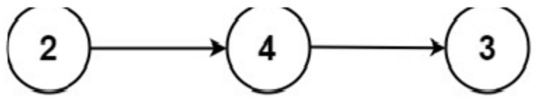
Input: head = [1, 2, 3, 4, 5], left = 2, right = 4
Output: [1, 4, 3, 2, 5]
Input: head = [5], left = 1, right = 1
Output: [5]
Input : [10,20,30,40,50,60,70], left = 3, right = 6
Output : [10,20,60,50,40,30,70]

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
	int data;
	struct Node* next;
};
int main() {
	int left, right;
	printf("Enter left and right positions: ");
	scanf("%d %d", &left, &right);
	if (left > right) {
```

```
            printf("Invalid input: left should be less than or equal to right.\n");
            return 1;      // Exit with an error code
    }
    // Create a sample linked list: 1 -> 2 -> 3 -> 4 -> 5
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 1;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 2;
    head->next->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->next->data = 3;
    head->next->next->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->next->next->data = 4;
    head->next->next->next->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->next->next->next->data = 5;
    head->next->next->next->next->next = NULL;
    struct Node* current = head;
    struct Node* prev = NULL;
    for (int i = 1; i < left; ++i) {
            prev = current;
            current = current->next;
    }
    struct Node* start = prev;
   struct Node* end = current;
   for (int i = left; i <= right; ++i) {
            struct Node* nextNode = current->next;
            current->next = prev;
            prev = current;
            current = nextNode;
    }
    if (start != NULL) {
            start->next = prev;
    } else {
            head = prev;
    }
    end->next = current;
    printf("Reversed list from position %d to %d: ", left, right);
    struct Node* printNode = head;
    while (printNode != NULL) {
            printf("%d ", printNode->data);
            printNode = printNode->next;
    }
    printf("\n");
    struct Node* temp;
    while (head != NULL) {
            temp = head;
            head = head->next;
            free(temp);
        }
        return 0;
}
```

29. 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.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
        int data;
        struct Node* next;
};
int main() {
        // Creating a linked list
        struct Node* head = malloc(sizeof(struct Node));
        head->data = 1;
        head->next = malloc(sizeof(struct Node));
        head->next->data = 2;
        head->next->next = malloc(sizeof(struct Node));
        head->next->next->data = 3;
        head->next->next->next = NULL;
        struct Node* current = head;
        while (current != NULL) {
                printf("%d ", current->data);
                current = current->next;
        }
        printf("\n");
        return 0;
}
```

30. 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

PROGRAM:

```
#include <stdio.h>
int main() {
        // Size of the arrays
        int m, n;
        printf("Enter the size of nums1: ");
        scanf("%d", &m);
        printf("Enter the size of nums2: ");
        scanf("%d", &n);
        int nums1[m], nums2[n];
        printf("Enter elements for nums1 in ascending order:\n");
        for (int i = 0; i < m; ++i) {
                scanf("%d", &nums1[i]);
        }
        printf("Enter elements for nums2 in ascending order:\n");
        for (int i = 0; i < n; ++i) {
                scanf("%d", &nums2[i]);
        }
        printf("Sum of the two sorted arrays: ");
                int i = 0, j = 0;
        while (i < m && j < n) {
                if (nums1[i] < nums2[j]) {
                        printf("%d ", nums1[i]);
                        i++;
                } else {
                        printf("%d ", nums2[j]);
                        j++;
                }
        }
        while (i < m) {
                printf("%d ", nums1[i]);
                i++;
        }
      while (j < n) {
                printf("%d ", nums2[j]);
                j++;
        }
```

```
    printf("\n");
    return 0;
}
```