LINEAR SEARCH

```c
#include <stdio.h>
int main()
{
    int size,i;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &size);
    int array[size];
    printf("Enter %d elements: ", size);
    for(i = 0;i<size;i++)
    {
        scanf("%d", &array[i]);
    }

    int target;
    printf("Enter the number to search: ");
    scanf("%d", &target);
    int index = -1;
    for(i=0;i<size;i++)
    {
        if(array[i]==target)
        {
            index = i;
            break;
        }
    }

    if(index!=-1)
    {
        printf("Element found at index %d\n", index);
    }
    else
    {
        printf("Element not found in the array\n");
    }

    return 0;
}
```

```
/tmp/Xqe08cpOv9.o
Enter the number of elements in the array: 5
Enter 5 elements: 11
12
13
14
15
Enter the number to search: 100
Element not found in the array


=== Code Execution Successful ===
```

BINARY SEARCH

```c
#include <stdio.h>
int main()
{
    int size;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &size);
    int array[size];
    printf("Enter %d sorted elements: ", size);
    for (int i = 0; i < size; i++)
    {
        scanf("%d", &array[i]);
    }
    int target;
    printf("Enter the number to search: ");
    scanf("%d", &target);
    int left=0;
    int right=size-1;
    int index=-1;
    while (left <= right)
    {
        int mid = left + (right - left) / 2;
        if(array[mid]==target)
        {
            index = mid;
            break;
        }
        if (array[mid] < target)
        {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    if (index != -1) {
        printf("Element found at index %d\n", index);
    } else {
        printf("Element not found in the array\n");
    }

    return 0;
}
```

```
/tmp/GhKWMFWlOc.o
Enter the number of elements in the array: 5
Enter 5 sorted elements: 10
20
30
40
50
Enter the number to search: 100
Element not found in the array


=== Code Execution Successful ===
```

INSERT AN ELEMENT IN ARRAY

```c
#include <stdio.h>
int main()
{
    int size, pos, element;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &size);
    int array[size + 1];
    printf("Enter %d elements: ", size);
    for (int i = 0; i < size; i++)
    {
        scanf("%d", &array[i]);
    }
    printf("Enter the position to insert the new element (0 to %d): ", size);
    scanf("%d", &pos);
    if (pos < 0 || pos > size)
    {
        printf("Invalid position!\n");
        return 1;
    }
    printf("Enter the element to insert: ");
    scanf("%d", &element);
    for (int i = size; i > pos; i--)
    {
        array[i] = array[i - 1];
    }
    array[pos] = element;
    printf("Array after insertion: ");
    for (int i = 0; i <= size; i++)
    {
        printf("%d ", array[i]);
    }
    printf("\n");
    return 0;
}

```

```
/tmp/Vo8NS1TBC.o
Enter the number of elements in the array: 5
Enter 5 elements: 123
14
15
13
16
Enter the position to insert the new element (0 to 5): 1
Enter the element to insert: 1
Array after insertion: 123 1 14 15 13 16


=== Code Execution Successful ===
```

DELETE ELEMENT IN ARRAY

```c
#include <stdio.h>
int main()
{
    int size, pos;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &size);
    int array[size];
    printf("Enter %d elements: ", size);
    for (int i = 0; i < size; i++)
    {
        scanf("%d", &array[i]);
    }
    printf("Enter the position to delete the element (0 to %d): ", size - 1);
    scanf("%d", &pos);
    if (pos < 0 || pos >= size)
    {
        printf("Invalid position!\n");
        return 1;
    }
    for (int i = pos; i < size - 1; i++)
    {
        array[i] = array[i + 1];
    }
    printf("Array after deletion: ");
    for (int i = 0; i < size - 1; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");

    return 0;
}
```

```
/tmp/JDc1WVvmnU.o
Enter the number of elements in the array: 5
Enter 5 elements: 24
22
23
26
28
Enter the position to delete the element (0 to 4): 2
Array after deletion: 24 22 26 28


=== Code Execution Successful ===
```

MERGE TWO ARRAY

```c
#include <stdio.h>
int main()
{
    int size1, size2;
    printf("Enter the number of elements in the first array: ");
    scanf("%d", &size1);
    int array1[size1];
    printf("Enter %d elements for the first array: ", size1);
    for (int i = 0; i < size1; i++)
    {
        scanf("%d", &array1[i]);
    }
    printf("Enter the number of elements in the second array: ");
    scanf("%d", &size2);
    int array2[size2];
    printf("Enter %d elements for the second array: ", size2);
    for (int i = 0; i < size2; i++)
    {
        scanf("%d", &array2[i]);
    }
    int mergedSize = size1 + size2;
    int mergedArray[mergedSize];
    for (int i = 0; i < size1; i++)
    {
        mergedArray[i] = array1[i];
    }

    for (int i = 0; i < size2; i++)
    {
        mergedArray[size1 + i] = array2[i];
    }

    printf("Merged array: ");
    for (int i = 0; i < mergedSize; i++)
    {
        printf("%d ", mergedArray[i]);
    }
    printf("\n");

    return 0;
}
```

```
/tmp/K03ICTRsLv.o
Enter the number of elements in the first array: 5
Enter 5 elements for the first array: 10
20
30
40
50
Enter the number of elements in the second array: 5
Enter 5 elements for the second array: 60
70
80
90
100
Merged array: 10 20 30 40 50 60 70 80 90 100


=== Code Execution Successful ===
```

Creating 4 nodes and Displaying

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};

int main()
{
    struct Node* head = NULL;
    struct Node* temp = NULL;
    struct Node* new_node = NULL;
    for (int i = 1; i <= 4; i++)
    {
        new_node = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i);
        scanf("%d", &new_node->data);
        new_node->next = NULL;

        if (head == NULL)
        {
            head = new_node;
            temp = head;
        }
        else
        {
            temp->next = new_node;
            temp = temp->next;
        }
    }
printf("\nLinked List elements:\n");
    temp = head;
    while (temp != NULL)
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
    temp = head;
    while (temp != NULL)
    {
        struct Node* next_node = temp->next;
```

```
/tmp/aZJpPJjF8O.o
Enter data for node 1: 25
Enter data for node 2: 35
Enter data for node 3: 45
Enter data for node 4: 55

Linked List elements:
25 -> 35 -> 45 -> 55 -> NULL


=== Code Execution Successful ===
```


singly linked list

```c
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int ele;
    struct node *next;
};

typedef struct node *LIST;
typedef LIST L;

L CreateNode()
{
    L temp;
    temp = (LIST)malloc(sizeof(LIST));
    temp->next = NULL;
    return temp;
}
void InsertAtEnd(L head, int num)
{
    L p = head;
    while (p->next != NULL)
        p = p->next;
    L temp;
    temp = (LIST)malloc(sizeof(LIST));
    temp->ele = num;
    p->next = temp;
    temp->next = NULL;
    // return temp;
}
L InsertAtBeginning(L head1, int num)
{
    L temp;
    temp = (LIST)malloc(sizeof(LIST));
    temp->ele = num;
    temp->next = head1;
    head1 = temp;
    return head1;
}
void InsertAfterElement(L head, int num, int a)
{
    L p = head;
```

```
/tmp/xURr1dVJC.o

Enter
1 - Insert Node
2 - Delete Node
3 - Display Node
0 - Exit
1

Enter the element to push : 26

Pushed
Enter
1 - Insert Node
2 - Delete Node
3 - Display Node
0 - Exit
3
26
Enter
1 - Insert Node
2 - Delete Node
3 - Display Node
0 - Exit
```

6th iteration

```c
#include <stdio.h>
void insertionSort(int arr[], int n)
{
    int i, j, key;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
     while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
        if (i == 6)
        {
            printf("Output:");
            for (int k = 0; k < n; k++)
            {
                printf("%d", arr[k]);
                if (k != n - 1)
                {
                    printf(",");
                }
            }
            printf("\n");
        }
    }
}
```

```
/tmp/qyvW6TOQ5I.o
Output:6,14,23,33,45,67,98,42
Output:6,14,23,33,42,45,67,98


=== Code Execution Successful ===
```

Fibonacci series with recursion

```c
#include <stdio.h>
int fibonacci(int n)
{
    if (n <= 1)
    {
        return n;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}
int fibonacciSeries(int n)
{
    int sum = 0;
    printf("Fibonacci Series up to %d terms:\n", n);
    for (int i = 0; i < n; i++)
    {
        int fib = fibonacci(i);
        sum += fib;
        printf("%d ", fib);
    }
    printf("\n");
    return sum;
}
int main()
{
    int n;
    printf("Enter the number of terms for Fibonacci series: ");
    scanf("%d", &n);

    int sum = fibonacciSeries(n);

    printf("Sum of Fibonacci series up to %d terms: %d\n", n, sum);

    return 0;
}
```

```
/tmp/fTPTnzKjmd.o
Enter the number of terms for Fibonacci series: 10
Fibonacci Series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
Sum of Fibonacci series up to 10 terms: 88


=== Code Execution Successful ===
```

Kth smallest

```c
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};
struct TreeNode* newNode(int val)
{
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = NULL;
    node->right = NULL;
    return node;
}

struct TreeNode* insert(struct TreeNode* node, int key)
{
    if (node == NULL)
        return newNode(key);

    if (key < node->val)
        node->left = insert(node->left, key);
    else if (key > node->val)
        node->right = insert(node->right, key);
    return node;
}
void kthSmallestUtil(struct TreeNode* root, int k, int* count, int* result)
{
    if (root == NULL || *count >= k)
        return;

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

```

```
/tmp/hWhXWwr6Ku.o
Enter the number of nodes in the BST: 5
Enter 5 elements for the BST:
10
30
40
50
20
Enter the value of k to find the kth smallest element: 2
Kth smallest element in BST: 20


=== Code Execution Successful ===
```

Frequency

```c
#include <stdio.h>
#include <string.h>
void findFrequency(char s[])
{
    int count[256] = {0};
    for (int i = 0; s[i] != '\0'; i++)
    {
        count[(unsigned char)s[i]]++;
    }

     printf("Character frequencies in the string:\n");
    for (int i = 0; i < 256; i++)
    {
        if (count[i] > 0)
        {
            printf("%c: %d\n", i, count[i]);
        }
    }
}

int main()
{
    char s[100];

    printf("Enter a string: ");
    scanf("%[^\n]", s);

    findFrequency(s);

    return 0;
}
```

```
/tmp/73qYzakyqA.o
Enter a string: data structures
Character frequencies in the string:
 : 1
a: 2
c: 1
d: 1
e: 1
r: 2
s: 2
t: 3
u: 2


=== Code Execution Successful ===
```

Bubble sort

```c
#include <stdio.h>
void bubbleSort(int arr[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (arr[j] < arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main()
{
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }
    printf("Original array: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
```

```
/tmp/PNvIMASvoB.o
Enter the number of elements: 5
Enter 5 elements:
20
14
10
5
2
Original array: 20 14 10 5 2
Sorted array in descending order: 20 14 10 5 2


=== Code Execution Successful ===
```

ODD- nodes

```c
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node* next;
};
void insert(struct Node** head, int newData)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = newData;
    newNode->next = *head;
    *head = newNode;
}
void findOddNumbers(struct Node* head)
{
    printf("Odd numbers present in the linked list:\n");
    while (head != NULL)
    {
        if (head->data % 2 != 0)
        {
            printf("%d\n", head->data);
        }
        head = head->next;
    }
}
int main()
{
    struct Node* head = NULL;
    int n, data;

    printf("Enter the number of nodes: ");
    scanf("%d", &n);
```

```
/tmp/EasJWUIaco.o
Enter the number of nodes: 4
Enter 4 integers to insert into the linked list:
1
2
3
7
Odd numbers present in the linked list:
7
3
1


=== Code Execution Successful ===
```

Needle Program

```c
#include <stdio.h>
#include <string.h>
int strStr(char* haystack, char* needle)
{
    int lenHaystack = strlen(haystack);
    int lenNeedle = strlen(needle);

    if (lenNeedle == 0)
        return 0;

    for (int i = 0; i <= lenHaystack - lenNeedle; i++)
    {
        int j;
        for (j = 0; j < lenNeedle; j++)
        {
            if (haystack[i + j] != needle[j])
                break;
        }
        if (j == lenNeedle)
            return i;
    }

    return -1;
}

int main()
{
    char haystack[100], needle[100];

    printf("Enter the haystack string: ");
    scanf("%s", haystack);

    printf("Enter the needle string: ");
    scanf("%s", needle);
```

```
/tmp/vJpOLIO3hZ.o
Enter the haystack string: leetcode
Enter the needle string: leeto
'leeto' is not found in 'leetcode'so -1


=== Code Execution Successful ===
```

Factorial using Recurision

```c
#include <stdio.h>
int factorial(int n)
{
    if (n == 0 || n == 1)
        return 1;
    else
        return n * factorial(n - 1);
}

int main()
{
    int num;

    printf("Enter a non-negative integer to compute its factorial: ");
    scanf("%d", &num);

    if (num < 0)
    {
        printf("Factorial is not defined for negative numbers.\n");
    }
    else
    {
        int result = factorial(num);
        printf("Factorial of %d is: %d\n", num, result);
    }

    return 0;
}

```

```
/tmp/BZ4I342u7r.o
Enter a non-negative integer to compute its factorial: 2
Factorial of 2 is: 2


=== Code Execution Successful ===
```

Repeated Character

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void sortString(char s[])
{
    int len = strlen(s);
    for (int i = 0; i < len - 1; i++)
    {
        for (int j = i + 1; j < len; j++)
        {
            if (tolower(s[i]) > tolower(s[j]))
            {
                char temp = s[i];
                s[i] = s[j];
                s[j] = temp;
            }
        }
    }
}
int findRepeatedCharStart(char s[])
{
    int len = strlen(s);
    for (int i = 0; i < len - 1; i++)
    {
        if (tolower(s[i]) == tolower(s[i + 1]))
        {
            return i;
        }
    }
    return -1;
}

int main()
{
```

```
/tmp/WPf8mlOoK9.o
Enter a string: tree
Sorted string in ascending order: eert
Starting index of repeated character 'e': 0


=== Code Execution Successful ===
```

Linked

```c
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node* next;
};

struct Node* insertNth(struct Node* head, int position, int newData)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = newData;
    newNode->next = NULL;

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

    struct Node* current = head;
    for (int i = 0; i < position - 1; i++)
    {
        if (current == NULL)
        {
            return head;
        }
        current = current->next;
    }

    if (current == NULL)
    {
        return head;
    }
```

```
/tmp/EKDcEYK7Xx.o
0 , 1


=== Code Execution Successful ===
```

C-Program For Heap Sort :

```c
#include <stdio.h>
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
void heapify(int arr[], int n, int i)
{
    int max = i;
    int leftChild = 2 * i + 1;
    int rightChild = 2 * i + 2;
    if (leftChild < n && arr[leftChild]>arr[max])
    {
        max = leftChild;
    }
    if (rightChild < n && arr[rightChild]> arr[max])
    {
        max = rightChild;
    }
    if (max != i)
    {
        swap(&arr[i], &arr[max]);
        heapify(arr, n, max);
    }
}
void heapSort(int arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
    {
        heapify(arr, n, i);
    }
    for (int i = n - 1; i >= 0; i--)
    {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}
void display(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
    {
```

```c
void display(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
    {
        printf("%d ", arr[i]);
    printf("\n");
    }
}
int main()
{
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; ++i)
    {
        scanf("%d", &arr[i]);
    }
    printf("Original array:\n");
    display(arr, n);
    heapSort(arr, n);
    printf("Sorted array:\n");
    display(arr, n);
    return 0;
}
```

Sample Input :

```
Enter number of elements: 5
Enter 5 integers:
20
15
336
69
70
```

Sample Output :

```
Original array:
20
15
336
69
70
Sorted array:
15
20
69
70
336


=== Code Execution Successful ===
```

C-Program For Stack implementation :

```c
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push()
{
    if(top>=n-1)
    {
        printf("\n\tSTACK is over flow");

    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK \n");
        for(i=top; i>=0; i--)
            printf("\n%d",stack[i]);
        printf("\n Press Next Choice");
    }
    else
    {
        printf("\n The STACK is empty");
    }

}
```

```c
int main()
{
    top=-1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d",&n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t--------------------------------");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
                break;
            }
            case 4:
            {
                printf("\n\t EXIT POINT ");
                break;
            }
            default:
            {
                printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
            }

        }
    }
    while(choice!=4);
    return 0;
}
```

Sample Input and Output:

```
Enter the size of STACK[MAX=100]:2

    STACK OPERATIONS USING ARRAY
   --------------------------------
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT
Enter the Choice:1
Enter a value to be pushed:3

Enter the Choice:1
Enter a value to be pushed:5

Enter the Choice:1

   STACK is over flow
Enter the Choice:3

The elements in STACK

5
3
Press Next Choice
Enter the Choice:
=== Session Ended. Please Run the code again ===
```

C-Program For Queue Implementation :

```c
#include<stdio.h>
#include<stdlib.h>
#define maxsize 5
void insert();
void delete();
void display();
int front = -1, rear = -1;
int queue[maxsize];
void main ()
{
    int choice;
    while(choice != 4)
    {
        printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");
        printf("\nEnter your choice :");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(0);
            break;
            default:
            printf("\nEnter valid choice??\n");
        }
    }
}
void insert()
{
```

```
{
    int item;
    printf("\nEnter the element\n");
    scanf("\n%d",&item);
    if(rear == maxsize-1)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    if(front == -1 && rear == -1)
    {
        front = 0;
        rear = 0;
    }
    else
    {
        rear = rear+1;
    }
    queue[rear] = item;
    printf("\nValue inserted ");

}
void delete()
{
    int item;
    if (front == -1 || front > rear)
    {
        printf("\nUNDERFLOW\n");
        return;
    }
    else
    {
        item = queue[front];
        if(front == rear)
        {
            front = -1;
            rear = -1 ;
        }
        else
        {
            front = front + 1;
        }
        printf("\nvalue deleted ");
    }
}
void display()
{
    int i;
    if(rear == -1)
    {
        printf("\nEmpty queue\n");
    }
    else
    {   printf("\nprinting values .....\n");
        for(i=front;i<=rear;i++)
        {
            printf("\n%d\n",queue[i]);
        }
    }
}
```

Sample input and output:

```
1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice :1

Enter the element
2

Value inserted
1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice :2

value deleted
1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice :3

Empty queue

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice :
```

12. Write a c program for AVL Tree.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
  int key;
  struct Node *left;
  struct Node *right;
  int height;
};
int max(int a, int b);
int height(struct Node *N) {
  if (N == NULL)
    return 0;
  return N->height;
}
int max(int a, int b) {
  return (a > b) ? a : b;
}
struct Node *newNode(int key) {
  struct Node *node = (struct Node *)
    malloc(sizeof(struct Node));
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  node->height = 1;
  return (node);
}
```

```
struct Node *rightRotate(struct Node *y) {
  struct Node *x = y->left;
  struct Node *T2 = x->right;
  x->right = y;
  y->left = T2;
  y->height = max(height(y->left), height(y->right)) + 1;
  x->height = max(height(x->left), height(x->right)) + 1;
  return x;
}
struct Node *leftRotate(struct Node *x) {
  struct Node *y = x->right;
  struct Node *T2 = y->left;
  y->left = x;
  x->right = T2;
  x->height = max(height(x->left), height(x->right)) + 1;
  y->height = max(height(y->left), height(y->right)) + 1;
  return y;
}
int getBalance(struct Node *N) {
  if (N == NULL)
    return 0;
  return height(N->left) - height(N->right);
}
struct Node *insertNode(struct Node *node, int key) {

  if (node == NULL)
    return (newNode(key));
```

```
  if (key < node->key)
    node->left = insertNode(node->left, key);
  else if (key > node->key)
    node->right = insertNode(node->right, key);
  else
    return node;
  node->height = 1 + max(height(node->left),
               height(node->right));
  int balance = getBalance(node);
  if (balance > 1 && key < node->left->key)
    return rightRotate(node);

  if (balance < -1 && key > node->right->key)
    return leftRotate(node);

  if (balance > 1 && key > node->left->key) {
    node->left = leftRotate(node->left);
    return rightRotate(node);
  }
  if (balance < -1 && key < node->right->key) {
    node->right = rightRotate(node->right);
    return leftRotate(node);
  }
  return node;
}
struct Node *minValueNode(struct Node *node) {
  struct Node *current = node;
```

```
  while (current->left != NULL)
    current = current->left;
  return current;
}
struct Node *deleteNode(struct Node *root, int key) {
  if (root == NULL)
    return root;
  if (key < root->key)
    root->left = deleteNode(root->left, key);
  else if (key > root->key)
    root->right = deleteNode(root->right, key);
  else {
    if ((root->left == NULL) || (root->right == NULL)) {
      struct Node *temp = root->left ? root->left : root->right;
      if (temp == NULL) {
        temp = root;
        root = NULL;
      } else
        *root = *temp;
      free(temp);
    } else {
      struct Node *temp = minValueNode(root->right);
      root->key = temp->key;
      root->right = deleteNode(root->right, temp->key);
    }
  }
  if (root == NULL)
```

```
  return root;
 root->height = 1 + max(height(root->left),
            height(root->right));
 int balance = getBalance(root);
 if (balance > 1 && getBalance(root->left) >= 0)
  return rightRotate(root);
 if (balance > 1 && getBalance(root->left) < 0) {
  root->left = leftRotate(root->left);
  return rightRotate(root);
 }
 if (balance < -1 && getBalance(root->right) <= 0)
  return leftRotate(root);
 if (balance < -1 && getBalance(root->right) > 0) {
  root->right = rightRotate(root->right);
  return leftRotate(root);
 }
 return root;
}
void printPreOrder(struct Node *root) {
 if (root != NULL) {
  printf("%d ", root->key);
  printPreOrder(root->left);
  printPreOrder(root->right);
 }
}
int main() {
 struct Node *root = NULL;
```

```
 root = insertNode(root, 2);
 root = insertNode(root, 1);
 root = insertNode(root, 7);
 root = insertNode(root, 4);
 root = insertNode(root, 5);
 root = insertNode(root, 3);
 root = insertNode(root, 8);
 printPreOrder(root);
 root = deleteNode(root, 3);
 printf("\nAfter deletion: ");
 printPreOrder(root);
 return 0;
}
```

Sample Input && Output :

```
Output
/tmp/yrBgoyQ3DQ.o
Enter the number of elements to insert: 5
Enter the elements: 25
36
78
12
11
Preorder traversal of the constructed AVL tree is:
36 12 11 25 78
Enter the element to delete: 12
Preorder traversal after deletion of 12:
36 25 11 78

=== Code Execution Successful ===
```

C-Program For Breadth first search:

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 50
typedef struct Graph_t
{
        int V;
        bool adj[MAX_VERTICES][MAX_VERTICES];
} Graph;
Graph* Graph_create(int V)
{
        Graph* g = malloc(sizeof(Graph));
        g->V = V;
        for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                        g->adj[i][j] = false;
                }
        }
        return g;
}
void Graph_destroy(Graph* g) { free(g); }
void Graph_addEdge(Graph* g, int v, int w)
```

```
{
        g->adj[v][w] = true;
}
void Graph_BFS(Graph* g, int s)
{
        bool visited[MAX_VERTICES];
        for (int i = 0; i < g->V; i++)
        {
                visited[i] = false;
        }
        int queue[MAX_VERTICES];
        int front = 0, rear = 0;
        visited[s] = true;
        queue[rear++] = s;

        while (front != rear)
        {
                s = queue[front++];
                printf("%d ", s);
                for (int adjacent = 0; adjacent < g->V;
                        adjacent++)
                        {
                        if (g->adj[s][adjacent] && !visited[adjacent])
```

```
                {
                    visited[adjacent] = true;
                    queue[rear++] = adjacent;
                }
            }
        }
}
int main()
{
    Graph* g = Graph_create(4);
    Graph_addEdge(g, 0, 1);
    Graph_addEdge(g, 0, 2);
    Graph_addEdge(g, 1, 2);
    Graph_addEdge(g, 2, 0);
    Graph_addEdge(g, 2, 3);
    Graph_addEdge(g, 3, 3);
    printf("Following is Breadth First Traversal "
           "(starting from vertex 2) \n");
    Graph_BFS(g, 2);
    Graph_destroy(g);
    return 0;
}
```

Sample Input :

```
Graph* g = Graph_create(4);
Graph_addEdge(g, 0, 1);
Graph_addEdge(g, 0, 2);
Graph_addEdge(g, 1, 2);
Graph_addEdge(g, 2, 0);
Graph_addEdge(g, 2, 3);
Graph_addEdge(g, 3, 3);
```

Sample Output :

Output

```
/tmp/glJVwI9vOF.o
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1

=== Code Execution Successful ===
```

C-Program To Traverse A Graph using DFS :

```
#include <stdio.h>
#include <stdlib.h>
struct node {
  int vertex;
  struct node* next;
};
struct node* createNode(int v);
struct Graph {
  int totalVertices;
  int* visited;
  struct node** adjLists;
};
void DFS(struct Graph* graph, int vertex) {
  struct node* adjList = graph->adjLists[vertex];
  struct node* temp = adjList;
  graph->visited[vertex] = 1;
  printf("%d -> ", vertex);
  while (temp != NULL) {
    int connectedVertex = temp->vertex;
    if (graph->visited[connectedVertex] == 0) {
      DFS(graph, connectedVertex);
    }
```

```
    temp = temp->next;
  }
}
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->totalVertices = vertices;
  graph->adjLists = malloc(vertices * sizeof(struct node*));
  graph->visited = malloc(vertices * sizeof(int));
  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}
void addEdge(struct Graph* graph, int src, int dest) {
  struct node* newNode = createNode(dest);
```

```
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}
void displayGraph(struct Graph* graph) {
  int v;
  for (v = 1; v < graph->totalVertices; v++) {
    struct node* temp = graph->adjLists[v];
    printf("\n%d =>  ", v);
    while (temp) {
      printf("%d, ", temp->vertex);
      temp = temp->next;
    }
    printf("\n");
  }
  printf("\n");
}
int main() {
  struct Graph* graph = createGraph(8);
  addEdge(graph, 1, 5);
  addEdge(graph, 1, 2);
```

```
  addEdge(graph, 1, 3);
  addEdge(graph, 3, 6);
  addEdge(graph, 2, 7);
  addEdge(graph, 2, 4);
  printf("\nThe Adjacency List of the Graph is:");
  displayGraph(graph);
  printf("\nDFS traversal of the graph: \n");
  DFS(graph, 1);
  return 0;
}
```

Sample Input :

```
addEdge(graph, 1, 5);
addEdge(graph, 1, 2);
addEdge(graph, 1, 3);
addEdge(graph, 3, 6);
addEdge(graph, 2, 7);
addEdge(graph, 2, 4);
```

Sample Output :

```
DFS traversal of the graph:
1 -> 3 -> 6 -> 2 -> 4 -> 7 -> 5 ->
```

C-Program Merge Sort :

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int l, int m, int r)
{
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;
        int L[n1], R[n2];
        for (i = 0; i < n1; i++)
          L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
          R[j] = arr[m + 1 + j];
        i = 0;
        j = 0;
        k = l;
        while (i < n1 && j < n2) {
          if (L[i] <= R[j]) {
                arr[k] = L[i];
                i++;
          }
          else {
```

```
                arr[k] = R[j];
                j++;
            }
            k++;
        }
        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }
        while (j < n2) {
            arr[k] = R[j];
            j++;
            k++;
        }
}
void mergeSort(int arr[], int l, int r)
{
        if (l < r) {
            int m = l + (r - l) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            merge(arr, l, m, r);
```

```
        }
}
void printArray(int A[], int size)
{
        int i;
        for (i = 0; i < size; i++)
          printf("%d ", A[i]);
        printf("\n");
}
int main()
{
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        int arr_size = sizeof(arr) / sizeof(arr[0]);
        printf("Given array is \n");
        printArray(arr, arr_size);
        mergeSort(arr, 0, arr_size - 1);
        printf("\nSorted array is \n");
        printArray(arr, arr_size);
        return 0;
}
```

Sample Input && sample output:

Output

```
/tmp/cXWrYSWzFb.o
Given array is
46 48 30 45 1

Sorted array is
1 30 45 46 48


=== Code Execution Successful ===
```

C-Program To Sort Elements using Quick Sort :

```c
#include<stdio.h>
void quicksort(int [10],int,int);
int main(){
  int x[20],size,i;
  printf("Enter size of the array: ");
  scanf("%d",&size);
  printf("Enter %d elements: ",size);
  for(i=0;i<size;i++)
    scanf("%d",&x[i]);
  quicksort(x,0,size-1);
  printf("Sorted elements: ");
  for(i=0;i<size;i++)
    printf(" %d",x[i]);
  return 0;
}
void quicksort(int x[10],int first,int last){
    int pivot,j,temp,i;
     if(first<last){
         pivot=first;
         i=first;
         j=last;
         while(i<j){
             while(x[i]<=x[pivot]&&i<last)
                 i++;
             while(x[j]>x[pivot])
                 j--;
             if(i<j){
                 temp=x[i];
                  x[i]=x[j];
                  x[j]=temp;
             }
         }
         temp=x[pivot];
         x[pivot]=x[j];
         x[j]=temp;
         quicksort(x,first,j-1);
         quicksort(x,j+1,last);
    }
}
```

Sample Input && Sample Output

```
Output
/tmp/6E4NHyhLhg.o
Enter the number of elements: 5
Enter the elements of the array: 45
60
1
2
5
Sorted Array
1 2 5 45 60

=== Code Execution Successful ===
```

Dijkstra’s Algorithm

```c
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>
#define V 9
int minDistance(int dist[], bool sptSet[])
{
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}

void printSolution(int dist[])
{
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src)
{
    int dist[V];
    bool sptSet[V];
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
    dist[src] = 0;
    for (int count = 0; count < V - 1; count++)
    {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;
        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }
    printSolution(dist);
}

int main()
{
    int graph[V][V];
    printf("Enter the adjacency matrix for the graph (9x9):\n");
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &graph[i][j]);
    int src;
    printf("Enter the source vertex: ");
    scanf("%d", &src);

    dijkstra(graph, src);

    return 0;
}
```

Sample input :

```
Enter the adjacency matrix for the graph (9x9):
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 14 10 0 2 0 0
0 0 0 0 0 2 0 1 6
8 11 0 0 0 0 1 0 7
0 0 2 0 0 0 6 7 0
```

SAMPLE OUTPUT

```
Enter the source vertex: 0
Vertex          Distance from Source
0               0
1               4
2               12
3               19
4               21
5               11
6               9
7               8
8               14


=== Code Execution Successful ===
```

PRIMS ALGORITHM

```c
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>
#define V 9
int minKey(int key[], bool mstSet[])
{
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;
    return min_index;
}
void printMST(int parent[], int graph[V][V])
{
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
}

void primMST(int graph[V][V])
{
    int parent[V];
    int key[V];
    bool mstSet[V];
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < V - 1; count++)
    {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }
    printMST(parent, graph);
}

int main()
{
    int graph[V][V];
    printf("Enter the adjacency matrix for the graph (9x9):\n");
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &graph[i][j]);

    primMST(graph);

    return 0;
}

```

Sample input

```
Output
/tmp/bkuUNatAVz.o
Enter the adjacency matrix for the graph (9x9):
0 2 0 6 0 0 0 0 0
2 0 3 8 5 0 0 0 0
0 3 0 0 7 0 0 0 0
6 8 0 0 9 0 0 0 0
0 5 7 9 0 0 0 0 0
0 0 0 0 0 0 1 2 3
0 0 0 0 0 1 0 4 0
0 0 0 0 0 2 4 0 0
0 0 0 0 0 3 0 0 0
```

Sample output

```
Edge    Weight
0 - 1    2
1 - 2    3
1 - 4    5
0 - 3    6
2 - 4    7
5 - 6    1
6 - 7    4
5 - 8    3
```

Kruskal Algorithm

```c
#include <stdio.h>
#include <stdlib.h>
#define V 9
struct Edge {
    int src, dest, weight;
};

struct Graph {
    int V, E;
    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
    graph->V = V;
    graph->E = E;
    graph->edge = (struct Edge*) malloc(graph->E * sizeof(struct Edge));
    return graph;
}

struct subset {
    int parent;
    int rank;
};

int find(struct subset subsets[], int i)
{
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);
    return subsets[i].parent;
}

void Union(struct subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;
    else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

int myComp(const void* a, const void* b)
{
    struct Edge* a1 = (struct Edge*)a;
    struct Edge* b1 = (struct Edge*)b;
    return a1->weight > b1->weight;
}

void printMST(struct Edge result[], int e)
{
    printf("Edge \tWeight\n");
    for (int i = 0; i < e; ++i)
```

Sample input

```
Output
/tmp/bkuUNatAVz.o
Enter the adjacency matrix for the graph (9x9):
0 2 0 6 0 0 0 0 0
2 0 3 8 5 0 0 0 0
0 3 0 0 7 0 0 0 0
6 8 0 0 9 0 0 0 0
0 5 7 9 0 0 0 0 0
0 0 0 0 0 0 1 2 3
0 0 0 0 0 1 0 4 0
0 0 0 0 0 2 4 0 0
0 0 0 0 0 3 0 0 0
```

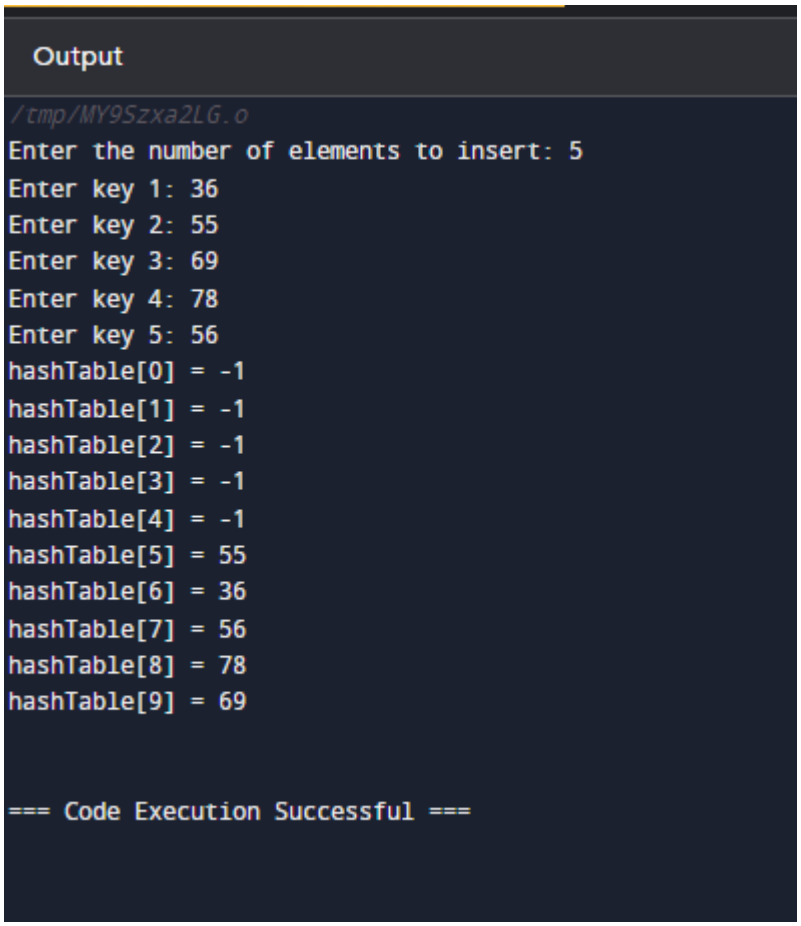
Sample Output

```
Edge      Weight
6 - 7      1
2 - 8      2
5 - 6      2
0 - 1      4
2 - 5      4
3 - 4      9
0 - 7      8
1 - 2      8
```

Linear Probing method:

```c
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10
int hashTable[TABLE_SIZE];
void initialize()
{
    for (int i = 0; i < TABLE_SIZE; i++)
        hashTable[i] = -1;
}
int hash(int key)
{
    return key % TABLE_SIZE;
}
void insert(int key)
{
    int index = hash(key);
    while (hashTable[index] != -1)
        index = (index + 1) % TABLE_SIZE;
    hashTable[index] = key;
}
void display()
{
    for (int i = 0; i < TABLE_SIZE; i++)
        printf("hashTable[%d] = %d\n", i, hashTable[i]);
}

int main()
{
    int n, key;
    printf("Enter the number of elements to insert: ");
    scanf("%d", &n);
    initialize();
    for (int i = 0; i < n; i++)
    {
        printf("Enter key %d: ", i + 1);
        scanf("%d", &key);
        insert(key);
    }
    display();
    return 0;
}

```

Sample input&&Sample output



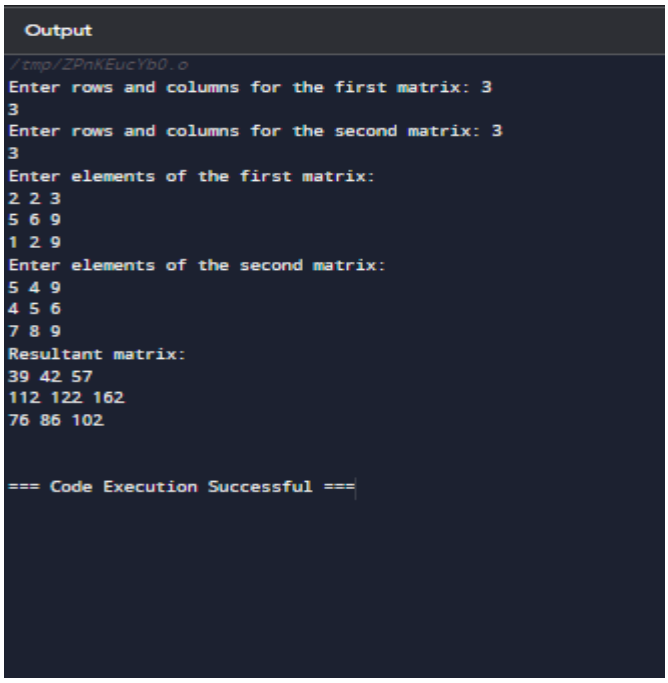
Output

```
/tmp/MY9Szxa2LG.o
Enter the number of elements to insert: 5
Enter key 1: 36
Enter key 2: 55
Enter key 3: 69
Enter key 4: 78
Enter key 5: 56
hashTable[0] = -1
hashTable[1] = -1
hashTable[2] = -1
hashTable[3] = -1
hashTable[4] = -1
hashTable[5] = 55
hashTable[6] = 36
hashTable[7] = 56
hashTable[8] = 78
hashTable[9] = 69


=== Code Execution Successful ===
```

Matrix Multiplication

```c
#include <stdio.h>
#include <stdlib.h>
void matrixMultiply(int r1, int c1, int r2, int c2, int mat1[][c1], int mat2[][c2], int res[][c2])
{
    for (int i = 0; i < r1; i++)
        for (int j = 0; j < c2; j++)
        {
            res[i][j] = 0;
            for (int k = 0; k < c1; k++)
                res[i][j] += mat1[i][k] * mat2[k][j];
        }
}
void printMatrix(int rows, int cols, int matrix[][cols])
{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
            printf("%d ", matrix[i][j]);
        printf("\n");
    }
}
int main()
{
    int r1, c1, r2, c2;
    printf("Enter rows and columns for the first matrix: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter rows and columns for the second matrix: ");
    scanf("%d %d", &r2, &c2);

    if (c1 != r2)
    {
        printf("Matrix multiplication is not possible.\n");
        return 0;
    }

    int mat1[r1][c1], mat2[r2][c2], res[r1][c2];

    printf("Enter elements of the first matrix:\n");
    for (int i = 0; i < r1; i++)
        for (int j = 0; j < c1; j++)
            scanf("%d", &mat1[i][j]);

    printf("Enter elements of the second matrix:\n");
    for (int i = 0; i < r2; i++)
        for (int j = 0; j < c2; j++)
            scanf("%d", &mat2[i][j]);

    matrixMultiply(r1, c1, r2, c2, mat1, mat2, res);

    printf("Resultant matrix:\n");
    printMatrix(r1, c2, res);

    return 0;
}

```

Sample input&&Sample output



```
Output
/tmp/ZPnKEucYb0.o
Enter rows and columns for the first matrix: 3
3
Enter rows and columns for the second matrix: 3
3
Enter elements of the first matrix:
2 2 3
5 6 9
1 2 9
Enter elements of the second matrix:
5 4 9
4 5 6
7 8 9
Resultant matrix:
39 42 57
112 122 162
76 86 102


=== Code Execution Successful ===
```

Application’s Of Stack

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
typedef struct
{
    int top;
    int items[MAX];
} Stack;

void initialize(Stack *s)
{
    s->top = -1;
}

int isEmpty(Stack *s)
{
    return s->top == -1;
}

int isFull(Stack *s)
{
    return s->top == MAX - 1;
}

void push(Stack *s, int value)
{
    if (!isFull(s))
        s->items[++(s->top)] = value;
}

int pop(Stack *s)
{
    if (!isEmpty(s))
        return s->items[(s->top)--];
    return -1;
}

int peek(Stack *s)
{
    if (!isEmpty(s))
        return s->items[s->top];
    return -1;
}

int precedence(char ch)
{
    switch (ch)
    {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
    }
```

Sample input && output

```
/tmp/oQXeOAswvt.o
Enter infix expression: 3+5*2
Postfix expression: 352*+
Evaluation result: 13


=== Code Execution Successful ===
```

Searching a MIN & MAX in BST

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

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

Node *insert(Node *root, int data)
{
    if (root == NULL)
        return createNode(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);
    return root;
}

Node *search(Node *root, int key)
{
    if (root == NULL || root->data == key)
        return root;
    if (root->data < key)
        return search(root->right, key);
    return search(root->left, key);
}

Node *findMin(Node *root)
{
    Node *current = root;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

Node *findMax(Node *root)
{
    Node *current = root;
    while (current && current->right != NULL)
        current = current->right;
    return current;
}

int main()
{
    Node *root = NULL;
    int n, data, key;

    printf("Enter the number of elements to insert into the BST: ");
    scanf("%d", &n);
```

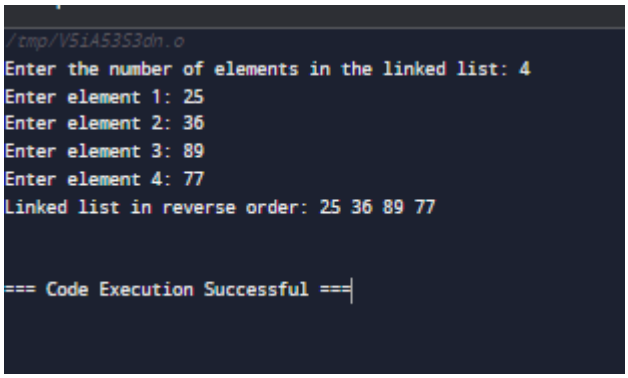
Sample input and output

```
/tmp/xW1DZiOjaV.o
Enter the number of elements to insert into the BST: 5
Enter element 1: 25
Enter element 2: 36
Enter element 3: 98
Enter element 4: 45
Enter element 5: 25
Enter the number to search in the BST: 98
98 is found in the BST.
Minimum value in the BST is: 25
Maximum value in the BST is: 98


=== Code Execution Successful ===
```

Reverse A linked list

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct Node
{
    int data;
    struct Node *next;
} Node;

Node *createNode(int data)
{
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insert(Node **head, int data)
{
    Node *newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

void printListReverse(Node *head)
{
    if (head == NULL)
        return;
    printListReverse(head->next);
    printf("%d ", head->data);
}

int main()
{
    Node *head = NULL;
    int n, data;

    printf("Enter the number of elements in the linked list: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
    {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &data);
        insert(&head, data);
    }

    printf("Linked list in reverse order: ");
    printListReverse(head);
    printf("\n");

    return 0;
}

```

Sample input & output



Sum of two arrays

```c
#include <stdio.h>
int sumArray(int arr[], int size)
{
    int sum = 0;
    for (int i = 0; i < size; i++)
        sum += arr[i];
    return sum;
}
int main()
{
    int m, n;
    printf("Enter the size of the first array: ");
    scanf("%d", &m);
    int nums1[m];
    printf("Enter elements of the first array:\n");
    for (int i = 0; i < m; i++)
    {
        printf("Element %d: ", i + 1);
        scanf("%d", &nums1[i]);
    }
    printf("Enter the size of the second array: ");
    scanf("%d", &n);
    int nums2[n];
    printf("Enter elements of the second array:\n");
    for (int i = 0; i < n; i++)
    {
        printf("Element %d: ", i + 1);
        scanf("%d", &nums2[i]);
    }
    int sum1 = sumArray(nums1, m);
    int sum2 = sumArray(nums2, n);
    int totalSum = sum1 + sum2;
    printf("The sum of the two arrays is: %d\n", totalSum);
    return 0;
}

```

Sample input & output

```
Enter the size of the first array: 5
Enter elements of the first array:
Element 1: 25
Element 2: 36
Element 3: 78
Element 4: 95
Element 5: 55
Enter the size of the second array: 2
Enter elements of the second array:
Element 1: 12
Element 2: 20
The sum of the two arrays is: 321


=== Code Execution Successful ===
```

Parenthesis validation

```c
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#define MAX 1000
typedef struct
{
    char items[MAX];
    int top;
} Stack;

void initialize(Stack* s)
{
    s->top = -1;
}

bool isEmpty(Stack* s) {
    return s->top == -1;
}

bool isFull(Stack* s) {
    return s->top == MAX - 1;
}

void push(Stack* s, char value) {
    if (!isFull(s)) {
        s->items[++(s->top)] = value;
    }
}

char pop(Stack* s) {
    if (!isEmpty(s)) {
        return s->items[(s->top)--];
    }
    return '\0';
}
char peek(Stack* s) {
    if (!isEmpty(s)) {
```

Sample input & output

```
Output

/tmp/fQE1460a68.o
Enter the string: ()[]{}
true


=== Code Execution Successful ===
```