LAB MANUAL

OF

DATA STRUCTURES AND ALGORITHM

B. TECH (CSE) 3rd SEMESTER

SUBJECT CODE - CSDC 231

SUBMITTED BY:

Name: Ritik Gupta

Roll Number: 23103122

GROUP: G2 CSE

SUBMITTED TO:

Name: Dr. Raj Mohan Singh

DEPARTMENT OFCOMPUTER SCIENCE AND ENGINEERING

DR. B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY JALANDHAR

DR B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY JALANDHAR

DEPARTMENT: COMPUTER SCIENCE AND ENGINEERING
COURSE CODE: CSDC-231
COURSE TITLE: DATA STRUCTURES AND ALGORITHM LABORATORY

LAB PRACTICALS (JULY-DEC, 2024)

S. No.	WEEK No.	NAME OF PRACTICAL	Date	Signature
1.	**WEEK 1**	Basic programs	6-08-24	
2.	**WEEK 2**	Arrays	13-08-24	
3.	**WEEK 3**	Pointers	27-08-24	
4.	**WEEK 4**	String	03-09-24	
5.	**WEEK 5**	Singly Linked list-01	10-09-24	
6.	**WEEK 6**	Singly Linked list-02	24-09-24	
7.	**WEEK 7**	Circular and Doubly Linked List	11-11-24	
8.	**WEEK 8**	Stacks and Queues	12-11-24	
9.	**WEEK 9**	Binary tree and AVL tree	19-11-24	
10.	**WEEK 10**			

WEEK-1

Basic Programs

1. Write a C program to perform addition, subtraction, multiplication, and division of two numbers entered by the user

```cpp
#include <iostream>
#include <math.h>
using namespace std;
void triangle(int a,int b,int c){
    if (((a*a)+(b*b)==(c*c)) ||((c*c)+(b*b)==(a*a)) || ((a*a)+(c*c)==(b*b)) )
    {
        cout<<"It is Right angle triangle.";
    }
    else if(a==b and b==c){
        cout<<"It is equilateral triangle";
    }
    else if((a==b) or (b==c) or (c==a)){
        cout<<"It is isosceles triangle";

    }
    else{
        cout<<"It is scalen triangle";
    }
}
int main(){
    int a,b,c;
    cout<<"Enter sides of triangle: ";
    cin>>a>>b>>c;
    triangle(a,b,c);
```

```
PS C:\Users\ritik\Desktop\lab\dsa> cd "c:\Users\ritik\Desktop\lab\dsa\Lab 1
Enter sides of triangle: 3 4 5
It is Right angle triangle.
```

2. Write a C program to check whether a given year is a leap year.

```cpp
#include <iostream>
using namespace std;

void triangle(int a,int b ,int c){
    int x =(a+b);
    int y = (b+c);
    int z = (c+a);
    if ((a>y) || (b>z) || (c>x) )
    {
        cout<<"Tringle is invalid";
    }
    else{
        cout<<"Triangle is valid";
    }
}
int main(){
    int a,b,c;
    cout<<"Enter sides of triangle: ";
    cin>>a>>b>>c;
    triangle(a,b,c);

PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ritik\Desktop\lab\dsa> cd "c:\Users\ritik\Desktop\la
Enter sides of triangle: 3 4 10
Tringle is invalid
```

3.Write a program to find quadrant.

```cpp
#include <iostream>
using namespace std;
void coordinate(int x ,int y)
{
    if(x>0 && y>0)
        cout<<"It is in 1st quadrant";
    else if(x>0 && y<0)
        cout<<"It is in 4th quadrant";
    else if(x<0 && y>0)
        cout<<"It is in 2nd quadrant";
    else if(x<0 && y<0)
        cout<<"It is in 3rd quadrant";
    else{

        cout<<"It is at origin";
    }
}
int main(){
    int x,y;
    cout<<"Enter coordiniates: ";
    cin>>x>>y;
    coordinate(x,y);

```

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ritik\Desktop\lab\dsa> cd "c:\Users\ritik\Desktop\lab
Enter coordiniates: -5 3
It is in 2nd quadrant
PS C:\Users\ritik\Desktop\lab\dsa\Lab 1>
```

4.Write a program to reverse a number.

```cpp
#include <iostream>
#include <math.h>
using namespace std;

int reverse_digit(int a){
    int temp = a;
    int num=0;
    for(int i=0;temp!=0;i++){
        int rem = temp%10;
        temp /= 10;
        num = (num*10) + rem;
    }
return num;
}
int main(){
    int a;
    cout<<"Kindly enter any number: ";
    cin>>a;
    cout<<"Reversed number is: "<<reverse_digit(a);
    return 0;
}
```

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ritik\Desktop\lab\dsa> cd "c:\Users\ritik\Desktop\lab\dsa\Lab 1\" ; if ($?) { g++ 12.c++ -o 12 } ; if ($?) { .\12 }
Kindly enter any number: 123
Reversed number is: 321
PS C:\Users\ritik\Desktop\lab\dsa\Lab 1>
```

5.Write a program tp find LCM of a number.

```cpp
#include<iostream>
using namespace std;

int lcm(int a,int b){
    int ans;
    for(int i=1;;i++){
        if ((i%a==0) && (i%b==0))
        {
            ans =i;
            break;
        }
    }
    return ans;
}

int main(){
    int a,b;
    cout<<"Enter any two number: ";
    cin>>a>>b;
    cout<<"LCM of these number is: "<<lcm(a,b);
    return 0;
}
```

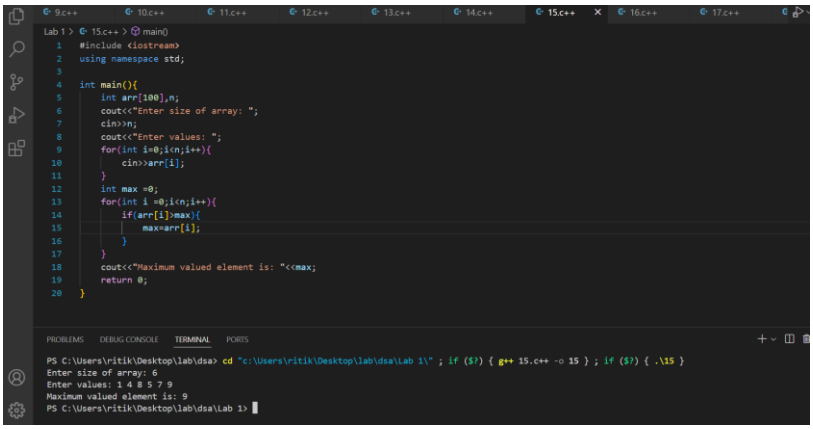
```
BLEMS  DEBUG CONSOLE  TERMINAL  PORTS

C:\Users\ritik\Desktop\lab\dsa> cd "c:\Users\ritik\Desktop\lab\dsa\Lab 1\" ; if ($?) { g++ 13.c++ -o 13 } ; if ($?) { .\13 }
er any two number: 5 6
 of these number is: 30
C:\Users\ritik\Desktop\lab\dsa\Lab 1>
```

6.Write a program to check weather a number is Armstrong or not.

```cpp
#include <iostream>
#include <math.h>
using namespace std;

int main(){
    int a,rem,size=0,sum=0,x;
    cout<<"Enter any number: ";
    cin>>a;

    int temp=a;
    for(int i=0;temp!=0;i++){
        rem = temp%10;
        temp /= 10;
        sum += (rem*rem*rem);
        size++;
    }
    if(sum==a){
        cout<<"Number is armstrong";
    }
    else
        cout<<"Number is not armstron";
    return 0;
}
```

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ritik\Desktop\lab\dsa> cd "c:\Users\ritik\Desktop\lab\dsa\Lab 1\" ; if ($?) { g++ 14.c++ -o 14 } ; if ($?) { .\14 }
Enter any number: 153
Number is armstrong
```

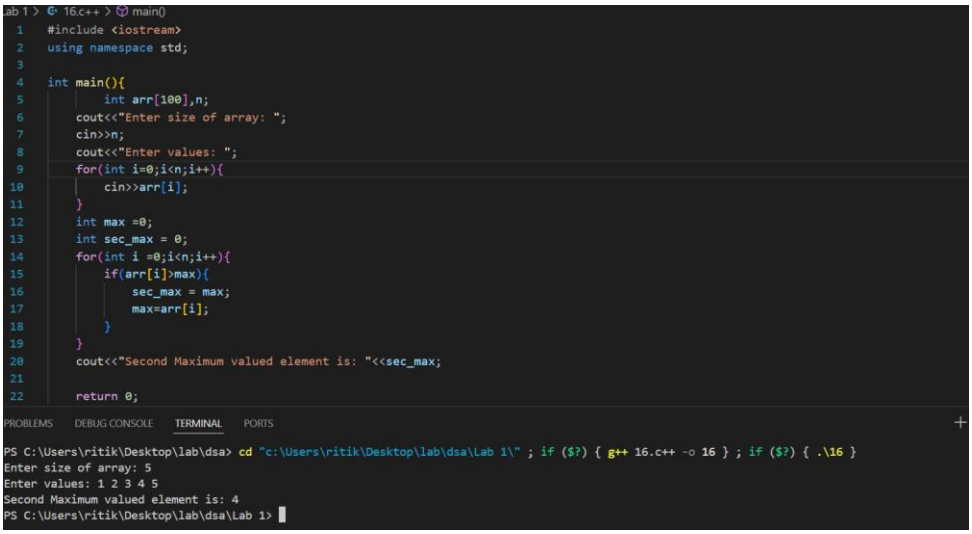
7.Write a program to find maximum vaalue in a array.



```
#include <iostream>
using namespace std;

int main(){
    int arr[100],n;
    cout<<"Enter size of array: ";
    cin>>n;
    cout<<"Enter values: ";
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int max =0;
    for(int i =0;i<n;i++){
        if(arr[i]>max){
            max=arr[i];
        }
    }
    cout<<"Maximum valued element is: "<<max;
    return 0;
}
```

```
PS C:\Users\ritik\Desktop\lab\dsa> cd "c:\Users\ritik\Desktop\lab\dsa\Lab 1\" ; if ($?) { g++ 15.c++ -o 15 } ; if ($?) { .\15 }
Enter size of array: 6
Enter values: 1 4 8 5 7 9
Maximum valued element is: 9
PS C:\Users\ritik\Desktop\lab\dsa\Lab 1>
```

8. Write a C program to check whether a given number is a prime number.



```
#include <iostream>
using namespace std;

int main(){
        int arr[100],n;
    cout<<"Enter size of array: ";
    cin>>n;
    cout<<"Enter values: ";
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int max =0;
    int sec_max = 0;
    for(int i =0;i<n;i++){
        if(arr[i]>max){
            sec_max = max;
            max=arr[i];
        }
    }
    cout<<"Second Maximum valued element is: "<<sec_max;

    return 0;
```

```
PS C:\Users\ritik\Desktop\lab\dsa> cd "c:\Users\ritik\Desktop\lab\dsa\Lab 1\" ; if ($?) { g++ 16.c++ -o 16 } ; if ($?) { .\16 }
Enter size of array: 5
Enter values: 1 2 3 4 5
Second Maximum valued element is: 4
PS C:\Users\ritik\Desktop\lab\dsa\Lab 1>
```

9. Write a C program to Check whether the triangle is equilateral, scalene, or isosceles USING function.

```
#include <iostream>
using namespace std;

int main(){
        int arr[100],n,dup[100],size=0;
    cout<<"Enter size of array: ";
    cin>>n;
    cout<<"Enter values: ";
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    cout<<"Duplicate values are: ";
    for(int i =0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(arr[i]==arr[j]){
                cout<<arr[i]<< " ";
            }
        }
    }
    return 0;
}
```

```
\Users\ritik\Desktop\lab\dsa> cd "c:\Users\ritik\Desktop\lab\dsa\Lab 1\" ; if ($?) { g++ 17.c++ -o 17 } ;
 size of array: 5
 values: 1 2 1 2 4
cate values are: 1 2
```

10. C program to check whether the triangle is valid or not if angles are given USING function.

```cpp
#include <iostream>
using namespace std;

int main(){
    int arr[100],n,sum=0;
    cout<<"Enter size of array: ";
    cin>>n;
    cout<<"Enter values: ";
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    for(int i =0;i<n;i++){
        sum += arr[i];
    }
    cout<<"Sum is: "<<sum;
    return 0;
}
```

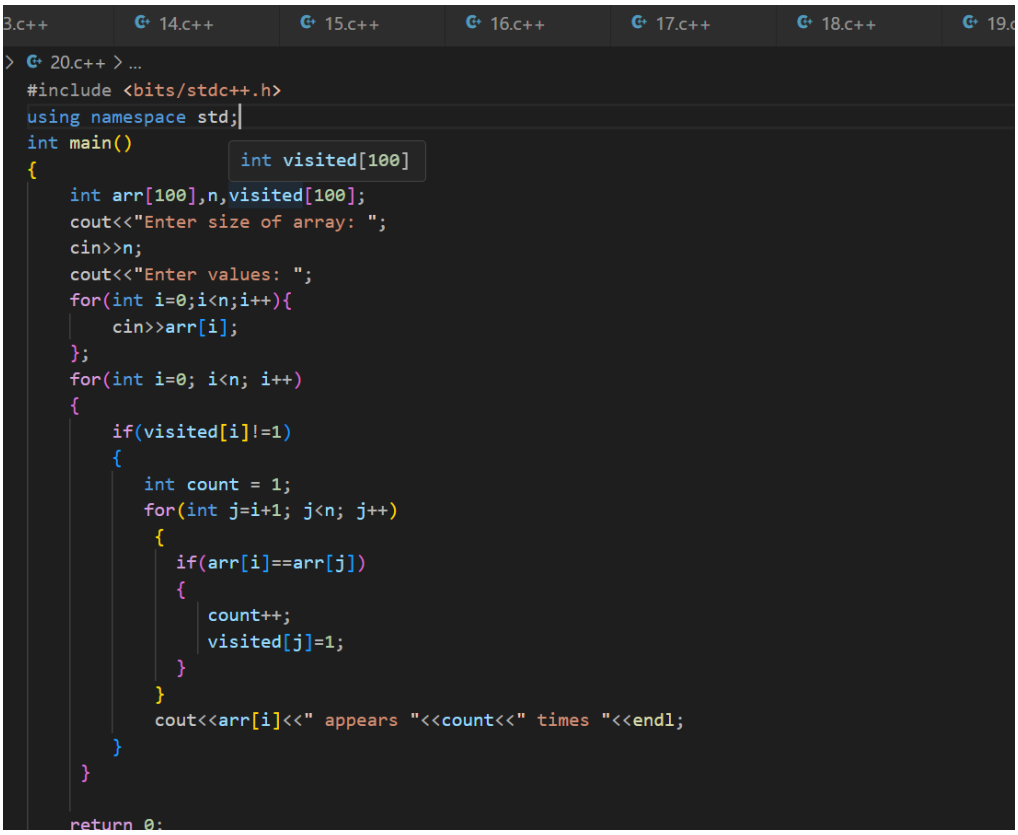
```
LEMS  DEBUG CONSOLE  TERMINAL  PORTS

:\Users\ritik\Desktop\lab\dsa> cd "c:\Users\ritik\Desktop\lab\dsa\Lab 1\" ; if ($?) { g++ 18.c++ -o 18 } ; if
er size of array: 6
er values: 1 4 5 2 3 6
is: 21
```

11. Write a C program to accept a coordinate point in an XY coordinate system and determine in which quadrant the coordinate point lies USING function.

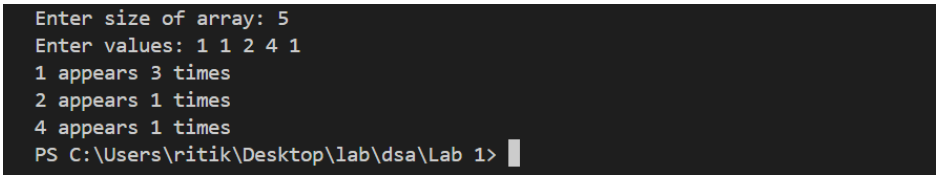
```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n;
    cout<<"Enter size: ";
    cin>>n;
    int Array[n];
    cout<<"Enter values: ";

    for(int i=0;i<n;i++){
        cin>>Array[i];
    }

    for(int i=0;i<n/2;i++){
        swap(Array[i],Array[n-1-i]);
    }

    cout<<"Reversed array: ";
    for(int i=0;i<n;i++){
        cout<<Array[i]<<" ";
    }

```

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ritik\Desktop\lab\dsa> cd "c:\Users\ritik\Desktop\lab\ds
Enter size: 5
Enter values: 1 7 4 5 6
Reversed array: 6 5 4 7 1
PS C:\Users\ritik\Desktop\lab\dsa\Lab 1>
```

12. Write a C program to reverse the digits of a given number.



```
3.c++  14.c++  15.c++  16.c++  17.c++  18.c++  19.c
> 20.c++ > ...
#include <bits/stdc++.h>
using namespace std;
int main()
{                    int visited[100]
    int arr[100],n,visited[100];
    cout<<"Enter size of array: ";
    cin>>n;
    cout<<"Enter values: ";
    for(int i=0;i<n;i++){
        cin>>arr[i];
    };
    for(int i=0; i<n; i++)
    {
        if(visited[i]!=1)
        {
           int count = 1;
           for(int j=i+1; j<n; j++)
            {
              if(arr[i]==arr[j])
              {
                  count++;
                  visited[j]=1;
              }
            }
            cout<<arr[i]<<" appears "<<count<<" times "<<endl;
        }
     }

    return 0;
```



```
Enter size of array: 5
Enter values: 1 1 2 4 1
1 appears 3 times
2 appears 1 times
4 appears 1 times
PS C:\Users\ritik\Desktop\lab\dsa\Lab 1>
```

******

WEEK-2

Arrays

1. Write a program to find the frequency of each element in an array.

```
int main()
{
    int arr[100],n,visited[100];
    cout<<"Enter size of array: ";
    cin>>n;
    cout<<"Enter values: ";
    for(int i=0;i<n;i++){
        cin>>arr[i];
    };
    for(int i=0; i<n; i++)
    {
        if(visited[i]!=1)
        {
           int count = 1;
           for(int j=i+1; j<n; j++)
            {
              if(arr[i]==arr[j])
              {
                 count++;
                 visited[j]=1;
              }
            }
            cout<<arr[i]<<" appears "<<count<<" times "<<endl;
        }
```

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ritik\Desktop\lab\dsa\lab2> cd "c:\Users\ritik\Desktop\lab\dsa
Enter size of array: 6
Enter values: 1 2 3 1 2 1
1 appears 3 times
2 appears 2 times
3 appears 1 times
PS C:\Users\ritik\Desktop\lab\dsa\lab2>
```

2.Write a program to find the common elements between two arrays.

```
#include <iostream>
using namespace std;
int main()
{
    int A[100],B[100],n,m,visited[100];
    cout<<"Enter size of array 1: ";
    cin>>n;
    cout<<"Enter values: ";
    for(int i=0;i<n;i++){
        cin>>A[i];
    };
    cout<<"Enter size of array 2: ";
    cin>>m;
    cout<<"Enter values: ";
    for(int i=0;i<m;i++){
        cin>>B[i];
    };
    cout<<"Common elements are: ";
    for(int i=0; i<n; i++){
        if(visited[i]!=1){
            for(int j=0; j<m; j++){
                if(A[i]==B[j]){
                    visited[j]=1;
                    cout<<A[i]<<"  ";
                }}}
    return 0;
```

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ritik\Desktop\lab\dsa\lab2> cd "c:\Users\ritik\D
Enter size of array 1: 4
Enter values: 1 2 3 4
Enter size of array 2: 5
Enter values: 7 8 9 1 3
Common elements are: 1  3
```

3. Write a program that uses a function to return multiple statistics (mean, median, mode) of an array.

```cpp
#include <bits/stdc++.h>
using namespace std;
int freq(int *arr,int n){
    int visited[100];
    int max=0;
    int mode=0;
    for(int i=0; i<n; i++){
        if(visited[i]!=1){
            int count = 1;
            for(int j=i+1;j<n; j++){
                if(arr[i]==arr[j]){
                    count++;
                    visited[j]=1;}}
             if(count>max){
                max=count;
                mode=arr[i];}}}
     return mode;}
void stats(int *arr,int n){
    int sum=0;
    // mean
    for(int i=0;i<n;i++){
        sum = sum+arr[i];
    }
    cout<<"Mean: "<<float(sum/2)<<endl;
    // median
    if(n%2==0){
        float median = (arr[n/2 -1]+arr[n/2])/2.0;
        cout<<"Median: "<<median<<endl;
    }
    else{
        int median=arr[(n+1)/2 - 1];
        cout<<"Median: "<<median<<endl;
    }
    // mode
    cout<<"Mode: "<<freq(arr,n);
}
int main()
{
    int arr[100],n;
    cout<<"Enter size of array: ";
    cin>>n;
    cout<<"Enter values: ";
    for(int i=0;i<n;i++){
        cin>>arr[i];
    };
    sort(arr,arr+n);
    stats(arr,n);
}
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab2> cd "c:\Use
Enter size of array: 6
Enter values: 1 2 4 1 5 3
Mean: 8
Median: 2.5
Mode: 1
PS C:\Users\ritik\Desktop\lab\dsa\lab2>
```

4. Write a program to perform scalar multiplication on a matrix.

```cpp
#include <iostream>
using namespace std;
int main()
{
    int arr[100][100],m,n,scal;
    cout<<"Enter row and column: ";
    cin>>m>>n;
    cout<<"Enter values: ";
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
        cin>>arr[i][j];
    };
    cout<<"Enter scaler: ";
    cin>>scal;
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
        arr[i][j]=2*arr[i][j];
    };
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
        cout<<arr[i][j]<< " ";
        }
        cout<<endl;
    };
}
```

```
Enter row and column: 3 3
Enter values: 4 5 6 1 8 7 9 2 4
Enter scaler: 2
8 10 12
2 16 14
18 4 8
```

5. Write a program to add/ subtract / multiply two matrices.

```cpp
#include <iostream>
using namespace std;
int main() {
    int A[100][100],B[100][100],C[100][100],m,n;
    cout<<"Enter row and column: ";
    cin>>m>>n;
    cout<<"Enter values in A: ";
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
        cin>>A[i][j];
    };
    cout<<"Enter values in B: ";
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
        cin>>B[i][j];
    };
    // Addition
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
        C[i][j]=A[i][j]+B[i][j];
    };
    cout<<"Added Matrix: ";
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
        cout<<C[i][j]<< " ";
```

```cpp
    // subtraction
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
        C[i][j]=A[i][j]-B[i][j];
    };
    cout<<"Subtracted Matrix: ";
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
        cout<<C[i][j]<< " ";}
        cout<<endl;
    }
    // Multiplication
    cout<<"Multiplied Matrix: ";
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            for (int k = 0; k < m; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            } }}
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            cout<<C[i][j]<< " ";}
        cout<<endl;}
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab2> cd "c:\
Enter row and column: 2 2
Enter values in A: 1 2 3 4
Enter values in B: 5 6 7 8
Added Matrix: 6 8
10 12
Subtracted Matrix: -4 -4
-4 -4
Multiplied Matrix: 15 18
39 46
```

6. Write a program to transpose a matrix.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int arr[100][100],tran[100][100],m,n;
    cout<<"Enter row and column: ";
    cin>>m>>n;
    cout<<"Enter values: ";
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
        cin>>arr[i][j];
    };
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
        tran[j][i]=arr[i][j];
    };
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
        cout<<tran[i][j]<<" ";
        }
        cout<<endl;
    };
}
```

```
PS C:\Users\ritik\Desktop\lab
Enter row and column: 2 2
Enter values: 1 2 3 4
1 3
2 4
```

7. Write a program to check if a matrix is symmetric/ equal:

```cpp
#include <iostream>
using namespace std;
int main()
{
    int arr[100][100],m,n,count=0;
    cout<<"Enter row and column: ";
    cin>>m>>n;
    cout<<"Enter values: ";
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
        cin>>arr[i][j];
    };
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
    if(arr[j][i]==arr[i][j]){
        count=2;}
    else{
    cout<<"Matrix is not symmetric";
    count=1;
    break;  }}
    if(count==1)
    break;
    }
    if(count==2)
    cout<<"Matrix is symmetric.";
}
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab2> cd "c:\Users\rit
Enter row and column: 3 3
Enter values: 1 2 3 2 5 6 3 6 9
Matrix is symmetric.
```

8. Write a program that finds the trace (sum of diagonal elements) of a matrix.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int arr[100][100],m,n,sum=0;
    cout<<"Enter row and column: ";
    cin>>m>>n;
    cout<<"Enter values: ";
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
        cin>>arr[i][j];
    };
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            if(i==j){
                sum = sum+arr[i][j];
            }
        }
    }
    cout<<"Trace is: "<<sum;
}
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab2> cd "c:\Users\ritik\Desktop\la
Enter row and column:  3 3
Enter values: 1 2 3 4 5 6 7 8 9
Trace is: 15
PS C:\Users\ritik\Desktop\lab\dsa\lab2>
```

9. Write a program to find the determinant of a 2x2 matrix.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int arr[100][100],m,n;
    cout<<"Enter values of square matrix: ";
    for(int i=0;i<2;i++){
        for(int j=0;j<2;j++)
        cin>>arr[i][j];
    };
   cout<<arr[0][1]<<" "<<arr[1][0]<<" "<<arr[1][1]<<endl;

    int det = ((arr[0][0]*arr[1][1]) - (arr[0][1]*arr[1][0]));
    cout<<"Determinant: "<<det;

}
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab2> cd "c:\Users\riti
Enter values of square matrix: 1 4 2 15
Determinant: 7
PS C:\Users\ritik\Desktop\lab\dsa\lab2>
```

*****

WEEK-3

POINTERS

1.Write a C program to declare an integer and a pointer to an integer. Assign the address of the integer to the pointer and use the pointer to modify the integer's value.

```cpp
#include <iostream>
using namespace std;

int main(){
    int a = 5;
    int *p = &a;
    *p = 10;
    cout<<a<<" "<<*p<<" "<<p<<" "<<&a;

    return 0;
}
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab3> cd '
 } ; if ($?) { .\1 }
10 10 0x61ff08 0x61ff08
PS C:\Users\ritik\Desktop\lab\dsa\lab3>
```

2. Create a C program that declares an array of integers. Use a pointer to traverse the array and print each element's value and address.

```cpp
#include <iostream>
using namespace std;

int main(){
    int arr[100] = {1,2,3,4,5};
    int n =5;
    int *p = arr;
    cout<<"Values are: "<<endl;
    for(int i =0;i<n;i++){
    cout<<*(p+i)<<" ";
    }
    cout<<endl;
    cout<<"Adresses are: "<<endl;
    for(int i =0;i<n;i++){
    cout<<(p+i)<<" ";
    }
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab3> cd "c:\Users\r
 } ; if ($?) { .\2 }
Values are:
1 2 3 4 5
Adresses are:
0x61fd70 0x61fd74 0x61fd78 0x61fd7c 0x61fd80
PS C:\Users\ritik\Desktop\lab\dsa\lab3>
```

3. Write a C program where you declare an integer, a pointer to an integer, and a pointer to a pointer to an integer. Initialize and display the values using these pointers.

```cpp
#include <iostream>
using namespace std;

int main(){
    int a = 5;
    int *p = &a;
    int **q = &p;

    cout<<a<<" "<<*p<<" "<<**q;
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab3> cd
 } ; if ($?) { .\3 }
5 5 5
PS C:\Users\ritik\Desktop\lab\dsa\lab3>
```

4. Write a function that takes two integer pointers as parameters and swaps the values they point to.Test this function in your main program.

```cpp
#include <iostream>
using namespace std;
void swap(int *a,int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
 }
int main(){
    int a ,b;
    cout<<"Enter a and b: ";
    cin>>a>>b;

    swap(&a,&b);
    cout<<"Swapped values are: "<<a<<" "<<b;

    return 0;
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab3> cd "c
  } ; if ($?) { .\4 }
 Enter a and b: 5 8
 Swapped values are: 8 5
PS C:\Users\ritik\Desktop\lab\dsa\lab3>
```

5. Implement a C program to find the maximum element in an array of integers using pointers.

```cpp
#include <iostream>
using namespace std;
int main(){
    int n,arr[100];
    cout<<"Enter size: ";
    cin>>n;
    cout<<"Enter elemnts: ";
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int *p =arr,max = arr[1];
    for(int i = 0;i<n;i++){
        if(*(p+i)>max){
            max = *(p+i);
        }
    }
    cout<<"Maximum element: "<<max;

    return 0;
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab3>
  } ; if ($?) { .\5 }
 Enter size: 5
 Enter elemnts: 8 5 7 4 9
 Maximum element: 9
PS C:\Users\ritik\Desktop\lab\dsa\lab3>
```

6. Write a C program that uses malloc to dynamically allocate an array of integers. Fill this array with data and then free the memory.

```cpp
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    int n;
    cout<<"Enter size: ";
    cin>>n;

    int* arr = (int*)malloc(n * sizeof(int));

    for (int i = 0; i < n; ++i) {
        arr[i] = i;
    }
    cout << "arr contents: " << endl;
    for (int i = 0; i < n; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;
    free(arr);
    return 0;
}
```

```
PS C:\Users\ritik\Desktop
Enter size: 5
arr contents:
0 1 2 3 4
```

7. Create a structure for a student that includes fields for ID and GPA. Write a function that takes a pointer to the student structure and prints the data.

```cpp
#include <iostream>
using namespace std;
struct student
{
    int ID;
    float GPA;
};
void print1(const student* stu){
    cout<<stu->ID<<endl;
    cout<<stu->GPA<<endl;
}
int main(){
    student ritik;
    ritik.ID = 122;
    ritik.GPA = 8.5;
    print1(&ritik);
```

```
PS C:\Users\ri
 } ; if ($?) {
122
8.5
PS C:\Users\ri
```

8. Write a function that takes an integer array and its size, and returns a pointer to a new array containing only even numbers from the input array.

```cpp
#include <iostream>
using namespace std;
int main(){
    int n,arr[100],even[100];
    cout<<"Enter size: ";
    cin>>n;
    cout<<"Enter values: ";
    for(int i=0;i<n;i++){
        cin>>arr[i];}
    int *p =arr;
    int *q = even;
    int count=0;
    for(int i=0;i<n;i++){
        if(arr[i]%2==0){
            *(q+count)=arr[i];
            count++;
        }}
    for(int i=0;i<count;i++){
        cout<<*(q+i)<<" ";
```

```
PS C:\Users\ritik\Desktop\lab\dsa\
  } ; if ($?) { .\8 }
Enter size: 6
Enter values: 1 4 8 6 2 9
4 8 6 2
```

9. Write a C program that reverses a string using pointers.

```cpp
#include <iostream>
#include<string>
using namespace std;

int main(){
    string str,str2 = "rit";
    cout<<"Enter string: ";
    cin>>str;
    char *p = &str[str.size()];

    for(int i=0;*p!='\0';i++){
        str2.append(*(p));
        cout<<i<<" ";
    }
    cout<<str2;
    return 0;
}
```

10. Write a program that creates an array of pointers to float. Use this array to store the addresses of five different float variables and print their values.

```cpp
#include <iostream>
using namespace std;
int main(){
    float *arr[100];
    float a = 5;
    float b = 3.2;
    float c = 2.7;
    float d = 6.9;
    float e = 8.8;
    arr[0] = &a;
    arr[1] = &b;
    arr[3] = &c;
    arr[4] = &d;
    arr[5] = &e;
    for (int i = 0; i < 5; ++i) {
        cout << "Value at floatArray[" << i << "] = " << *arr[i] <<endl;
    }
    return 0;
}
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab3> cd
10 } ; if ($?) { .\10 }
Value at floatArray[0] = 5
Value at floatArray[1] = 3.2
PS C:\Users\ritik\Desktop\lab\dsa\lab3>
```

11. Implement a C program that has a function to calculate the sum and average of an array of integers. Pass the array and its size to the function using pointers.

```
#include <iostream>
using namespace std;
void sumandavg(int *arr,int n){
   int sum =0;
   for(int i=0;i<n;i++){
    sum +=arr[i];
   }
   float avg = sum/n;
   cout<<"Sum is: "<<sum<<endl;
   cout<<"Average is: "<<avg;}
int main(){
    int n ,arr[100];
    cout<<"Enter size: ";
    cin>>n;
    cout<<"Enter values: "<<endl;
    for(int i =0;i<n;i++){
        cin>>arr[i];}
    sumandavg(arr,n);
```

```
11 } ; if ($?) { .\11 }
Enter size: 5
Enter values:
4 5 6 7 8
Sum is: 30
Average is: 6
```

12. Write a C program that takes command line arguments and counts the total number of characters in all the arguments using pointers.

```
using namespace std;
#include <iostream>

int countTotalCharacters(int argc, char* argv[]) {
    int totalCount = 0;
    for (int i = 1; i < argc; ++i) {
        char* arg = argv[i];
        while (*arg) {
            ++totalCount;
            ++arg;
        }
    }
    return totalCount;
}
int main(int argc, char* argv[]) {
    int totalCharacters = countTotalCharacters(argc, argv);
    cout<< "Total number of characters in all arguments: " << totalCharacters <<endl;
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab3> cd "c:\Users\ritik\De
12 } ; if ($?) { .\12 }
Total number of characters in all arguments: 0
PS C:\Users\ritik\Desktop\lab\dsa\lab3> .\12 ritik gupta
Total number of characters in all arguments: 10
PS C:\Users\ritik\Desktop\lab\dsa\lab3>
```

13. Create a function that takes three integers as input and uses pointers to return their sum, average, and product.

```cpp
#include <iostream>
using namespace std;

int main(){
 int a ,b,c;

 cout<<"Enter three numbers: ";
 cin>>a>>b>>c;
 int *p = &a;
 int *q = &b;
 int *r = &c;
 int sum = *p+*q+*r;
 float avg = sum/2;

cout<<"Sum is: "<<sum<<endl;
cout<<"Avg is: "<<avg;
    return 0;
}
```

```
PS C:\Users\ritik\Desktop\lab\ds
13 } ; if ($?) { .\13 }
Enter three numbers: 4 9 7
Sum is: 20
Avg is: 10
PS C:\Users\ritik\Desktop\lab\ds
```

14. Write a program that declares a pointer to a function which takes two integers and returns a float. Implement and test the function.

```cpp
#include <iostream>
using namespace std;

float divide(int a, int b) {
    return static_cast<float>(a) / b;
}

int main() {
    float (*funcPtr)(int, int);
    funcPtr = &divide;
    int num1,num2;
    cout<<"Enter two numbers: ";
    float result = funcPtr(num1, num2);
    cout << "Result of " << num1 << " / " << num2 << " = " << result << endl;

    return 0;
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab3>
14 } ; if ($?) { .\14 }
Enter two numbers: 48 12
Result of 48 / 12 = 4
PS C:\Users\ritik\Desktop\lab\dsa\lab3>
```

15. Develop a C program that dynamically allocates memory for a 2D array using pointers. Fill the array with values and then print them.

```cpp
#include <iostream>
using namespace std;

int main() {
    int rows, cols;
    cout << "Enter the number of rows: ";
    cin >> rows;
    cout << "Enter the number of columns: ";
    cin >> cols;

    int** array = new int*[rows];
    for (int i = 0; i < rows; i++) {
        array[i] = new int[cols];
    }
    cout << "Filling the array with values:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            array[i][j] = i * cols + j;       }}
    cout << "\nThe array values are:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << array[i][j] << " "; }
        cout << endl;}
    for (int i = 0; i < rows; i++) {
        delete[] array[i]; }
    delete[] array;
    return 0;
}
```

```
PS C:\Users\ritik\Desktop\lab\dsa\lab3> cd "c:\Users
Enter the number of rows: 5
Enter the number of columns: 3
Filling the array with values:

The array values are:
0 1 2
3 4 5
6 7 8
9 10 11
12 13 14
PS C:\Users\ritik\Desktop\lab\dsa\lab3>
```

******

WEEK-04

STRINGS

1. Write a C program to find length of a string.

```cpp
#include <iostream>
#include<string.h>
using namespace std;

int main() {
   string str;
   int length =0;

   cout << "Enter a string: ";
   cin>>str;

   for(int i=0;str[i]!='\0';i++){
      length++;
   }

   cout << "Length of the string: " << length << endl;

   return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\C
Enter a string: ritikgupta
Length of the string: 10
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

2. Write a C program to copy one string to another string.

```cpp
#include <iostream>
using namespace std;
#include <cstring>

int main() {
   string source, destination;
   cout << "Enter the source string: ";
   cin>>source;

      destination=source;
```

```
    cout << "Source string: " << source << endl;
    cout << "Destination string: " << destination << endl;

    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\OneDri
Enter the source string: ritikgupta
Source string: ritikgupta
Destination string: ritikgupta
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

3. Write a C program to concatenate two strings.

```
#include <iostream>
using namespace std;
#include <cstring>

int main() {
    string str1, str2;
    cout << "Enter the first string: ";
    cin>>str1;

    cout << "Enter the second string: ";
    cin>>str2;

    str1 = str1 + str2;

    cout << "Concatenated string: " << str1 << endl;

    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ri
Enter the first string: ritik
Enter the second string: gupta
Concatenated string: ritikgupta
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

4. Write a C program to compare two strings.

```
#include <iostream>
```

```cpp
#include <string>
using namespace std;

int main() {
   string str1, str2;
   cout << "Enter the first string: ";
   getline(cin, str1);
   cout << "Enter the second string: ";
   getline(cin, str2);
   bool areEqual = true;
   int minLength = min(str1.length(), str2.length());
   for (int i = 0; i < minLength; i++) {
      if (str1[i] != str2[i]) {
         areEqual = false;
         if (str1[i] < str2[i]) {
            cout << "The first string is less than the second string." << endl;}
         else {
            cout << "The first string is greater than the second string." << endl;}
         break;
         }}
   if (areEqual) {
      if (str1.length() == str2.length()) {
         cout << "The strings are equal." << endl;
      } else if (str1.length() < str2.length()) {
         cout << "The first string is less than the second string." << endl;
      } else {
         cout << "The first string is greater than the second string." << endl;
      }}
   return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\O
Enter the first string: ritik
Enter the second string: ritika
The first string is less than the second string.
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

5. Write a C program to find reverse of a string.

```cpp
#include <iostream>
#include <string>
using namespace std;
```

```
int main(){
   string str;
   cout<<"Enter string: ";
   getline(cin,str);
   int st = 0;
   int end = str.length()-1;

   while(st<=end){
      swap(str[st],str[end]);
      st++;
      end--;
   }

   cout<<"Reversed stirng: "<<str;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\On
Enter string: ritik gupta
Reversed stirng:  atpug kitir
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

7. Write a C program to check whether a string is palindrome or not.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
   string str;
   cout << "Enter a string: ";
   getline(cin, str);
   bool palindrome = true;
   for(int i=0;i<str.length();i++){
      if(str[i]!=str[str.length()-i-1]){
         palindrome = false;
         break;
      }
   }
   if(palindrome){
      cout<<"String is palindrome.";
   }
   else
      cout<<"String is not palindrome.";
```

```
  return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\OneDrive\Desktop\la
Enter a string: ritikitir
String is palindrome.
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\OneDrive\Desktop\la
Enter a string: ritikgupta
String is not palindrome.
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

8. Write a C program to find highest frequency character in a string.

```
#include <iostream>
using namespace std;
#include <map>
int main(){
 map<char,int>m;
 string str;

cout<<"Enter string: ";
getline(cin,str);

for(int i=0;i<str.length();i++){
   m[str[i]]++;
}
char ans;
int count =0;
for(int i=0;i<str.length();i++){
   if(m[str[i]]>count){
      count=m[str[i]];
      ans = str[i];
   }
}
cout<<"Max frequent characther "<<ans<<" has frequency "<<count;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\
Enter string: ritik is a nice and rigid boy
Max frequent characther i has frequency 6
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

9. Write a C program to convert a user-input string to uppercase.

```
#include<bits/stdc++.h>
```

```cpp
using namespace std;

int main(){
  string str;
  cout<<"Enter string: ";
  getline(cin,str);
  transform(str.begin(), str.end(), str.begin(), ::toupper);

  cout<<"Uppercsed: "<<str;
 }
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:
Enter string: ritIKgupTa
Uppercsed: RITIKGUPTA
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

10. Create a C program where the user enters two strings, and the program checks if the second string is a substring of the first string.

```cpp
#include<bits/stdc++.h>
using namespace std;
map<char,int> m1;
map<char,int> m2;

int main(){
  string str1 ,str2;
  cout<<"Enter string 1: ";
  getline(cin,str1);
  cout<<"Enter string 1: ";
  getline(cin,str2);

  for(int i =0;i<str1.length();i++){
   m1[str1[i]]++;
  }
  for(int j=0;j<str2.length();j++){
   m2[str2[j]]++;
   }
   bool ans = true;
   for(int i=0;i<str2.length();i++){
      if(m2[str2[i]]>m1[str2[i]]){
         ans = false;
         break;
       }
```

```
  }
  if(ans){
     cout<<"String 2 is substring of 1 ";
  }
  else
     cout<<"String 2 is not a substring of 1 ";
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\OneD
Enter string 1: ritikgupta
Enter string 1: tikgu
String 2 is substring of 1
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

11. Implement a C program that reads a sentence from the user and counts the number of words in the sentence.

```
#include<bits/stdc++.h>
using namespace std;

int main(){
  string str;
  cout<<"Enter string: ";
  getline(cin,str);
  int count=1;
  for(int i =0;i<str.length();i++){
     if(str[i] == ' '){
        count++;
     }
  }
  cout<<"Number of words: "<<count;
  }
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Use
Enter string: DSA is my favourite subject
Number of words: 5
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

12. Write a C program that takes a sentence as input and displays each word on a new line.

```
#include<bits/stdc++.h>
using namespace std;
```

```cpp
int main(){
  string str;
  cout<<"Enter string: ";
  getline(cin,str);

  for(int i =0;i<str.length();i++){
   if(str[i]==' '){
      cout<<endl;
   }
   else
   cout<<str[i];
  }
  }
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\
Enter string: DSA is easy
DSA
is
easy
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

13. Write a C program that asks the user for a string and then prints the frequency of each character that appears in the string.

```cpp
#include<bits/stdc++.h>
using namespace std;
int main(){
  string str;
  map<char,int>m;
  cout<<"Enter string: ";
  getline(cin,str);
for(int i =0;i<str.length();i++){
   m[str[i]]++;
  }
for(int i=0;i<str.length();i++){
   if(m[str[i]]!=0  && str[i] != ' '){
   cout<<str[i]<<" repeats "<<m[str[i]]<< " times"<<endl;
   m[str[i]]=0;}
}
  }
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4\" ; if ($?) {
Enter string: dsa is nice but dsa is tough subject
d repeats 2 times
s repeats 5 times
a repeats 2 times
i repeats 3 times
n repeats 1 times
c repeats 2 times
e repeats 2 times
b repeats 2 times
u repeats 3 times
t repeats 3 times
o repeats 1 times
g repeats 1 times
h repeats 1 times
j repeats 1 times
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

14. Create a C program that reads a sentence from the user and finds the longest word in the sentence. Print the longest word and its length.

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
  string str;
  cout<<"Enter string: ";
  getline(cin,str);
  int count=0;
  int ans =0;
  for(int  i =0;i<str.length();i++){
      if(str[i]==' '){
         if(count>ans){
         ans =count;}
         count=0;
      }
      else{
         count++;
      }
  }
  cout<<"Max length: "<<ans;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\OneDriv
Enter string: ritik gupta
Max length: 5
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

15. Write a C program that allows the user to input a string, a character to replace, and a character with which to replace it.

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    string str;
    char toReplace, replaceWith;
    cout << "Enter a string: ";
    getline(cin, str);
    cout << "Enter character to replace: ";
    cin >> toReplace;
    cout << "Enter replacement character: ";
    cin >> replaceWith;
    for (char &c : str) {
        if (c == toReplace) c = replaceWith;
    }
    cout << "Modified string: " << str;
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\OneDrive\Desktop\lab\dsa\la
Enter a string: ritika gupta
Enter character to replace: a
Enter replacement character: g
Modified string: ritikg guptg
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

16. Write a program that sorts the characters in a string alphabetically.

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    string str;
    cout << "Enter a string: ";
    cin >> str;
    int n = str.length();
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (str[j] > str[j + 1]) {
                swap(str[j], str[j + 1]);
            }
        }
    }
    cout << "Sorted string: " << str;
```

```
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\OneD
Enter a string: gupta
Sorted string: agptu
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

17. Write a program to Print and remove duplicate characters from a string.

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string str;
    cout << "Enter a string: ";
    cin >> str;
    string result;
    bool isDuplicate;
    for (char c : str) {
        isDuplicate = false;
        for (char r : result) {
            if (c == r) {
                isDuplicate = true;
                break;
            }
        }
        if (!isDuplicate) {
            result += c;
        }
    }
    cout << "String without duplicates: " << result;
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\OneD
Enter a string: cse is nice but cse has more strenght
String without duplicates: cse
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

18. Write a program to check if two strings are anagrams of each other.

```cpp
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;
int main() {
    string str1, str2;
    cout << "Enter first string: ";
    cin >> str1;
    cout << "Enter second string: ";
    cin >> str2;
    if (str1.length() != str2.length()) {
        cout << "No, they are not anagrams.";
        return 0;
    }
    unordered_map<char, int> charCount;
    for (char c : str1) {
        charCount[c]++;
    }
    for (char c : str2) {
        charCount[c]--;
        if (charCount[c] < 0) {
            cout << "No, they are not anagrams.";
            return 0;
        }
    }
    cout << "Yes, they are anagrams.";
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\One
Enter first string: ritik
Enter second string: tikri
Yes, they are anagrams.
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

19. Write a program to find the first non-repeating character and position in a string.

```cpp
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;
int main() {
```

```
    string str;
    cout << "Enter a string: ";
    cin >> str;
    unordered_map<char, int> count;
    for (char c : str) count[c]++;
    for (int i = 0; i < str.size(); i++) {
        if (count[str[i]] == 1) {
            cout << "First non-repeating character: " << str[i] << " at position " << i;
            return 0;
        }
    }
    cout << "No non-repeating character found.";
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4> cd "c:\Users\ritik\
Enter a string: ritikgupta
First non-repeating character: r at position 0
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 4>
```

20. Write a program to find the longest palindromic substring in a given string.

```
#include <iostream>
#include <string>
using namespace std;
bool isPalindrome(const string &s, int start, int end) {
    while (start < end) {
        if (s[start++] != s[end--]) return false;
    }
    return true;
}
int main() {
    string str;
    cout << "Enter a string: ";
    cin >> str;
    int maxLength = 1, start = 0;
    for (int i = 0; i < str.length(); i++) {
        for (int j = i; j < str.length(); j++) {
            if (isPalindrome(str, i, j)) {
                int length = j - i + 1;
                if (length > maxLength) {
                    maxLength = length;
```

```
                start = i;
            }
        }
    }
}
cout << "Longest palindromic substring: " << str.substr(start, maxLength);
return 0;
}
```

******

WEEK – 05

LINKED LIST-01

1. Write a program to implement a singly linked list and its basic operations like

a. insertion,

I. at the beginning

II. at the end

III. at any specific location

b. deletion,

I. at the beginning

II. at the end

III. at any specific location

c. Searching

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
Node* head = nullptr;
void insertAtBeginning(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}
void insertAtEnd(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
```

```
        temp->next = newNode;
    }
}
void insertAtPosition(int value, int pos) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    if (pos == 1) {
        newNode->next = head;
        head = newNode;
        return;
    }
    Node* temp = head;
    for (int i = 1; i < pos - 1; i++) {
        if (temp == nullptr) return;
        temp = temp->next;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}
void deleteAtBeginning() {
    if (head == nullptr) return;
    Node* temp = head;
    head = head->next;
    delete temp;
}
void deleteAtEnd() {
    if (head == nullptr) return;
    if (head->next == nullptr) {
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    while (temp->next->next != nullptr) {
        temp = temp->next;
    }
    delete temp->next;
    temp->next = nullptr;
}
void deleteAtPosition(int pos) {
    if (head == nullptr) return;
    if (pos == 1) {
        Node* temp = head;
        head = head->next;
```

```
        delete temp;
        return;
    }
    Node* temp = head;
    for (int i = 1; i < pos - 1; i++) {
        if (temp == nullptr || temp->next == nullptr) return;
        temp = temp->next;
    }
    Node* delNode = temp->next;
    temp->next = temp->next->next;
    delete delNode;
}
bool search(int value) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == value) return true;
        temp = temp->next;
    }
    return false;
}
void print() {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    int value, pos;
    cout << "Enter a value to insert at the beginning: ";
    cin >> value;
    insertAtBeginning(value);
    print();
    cout << "Enter a value to insert at the end: ";
    cin >> value;
    insertAtEnd(value);
    print();
    cout << "Enter a value to insert and position: ";
    cin >> value >> pos;
    insertAtPosition(value, pos);
    print();
    cout << "Deleting node at the beginning...\n";
    deleteAtBeginning();
    print();
```

```
    cout << "Deleting node at the end...\n";
    deleteAtEnd();
    print();
    cout << "Enter the position of the node to delete: ";
    cin >> pos;
    deleteAtPosition(pos);
    print();
    cout << "Enter a value to search: ";
    cin >> value;
    if (search(value)) {
        cout << "Value found.\n";
    } else {
        cout << "Value not found.\n";
    }
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 5> cd "c:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 5\" ; if ($?) { g++ 1.c++ -o 1 } ; if ($?) { .
Enter a value to insert at the beginning: 10
10
Enter a value to insert at the end: 30
10 30
Enter a value to insert and position: 20 2
10 20 30
Enter a value to insert at the end: 40
10 20 30 40
Enter a value to insert at the end: 50
10 20 30 40 50
Deleting node at the beginning...
20 30 40 50
Deleting node at the end...
20 30 40
Enter the position of the node to delete: 2
20 40
Enter a value to search: 40
Value found.
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 5>
```

2. Write a program to reverse a singly linked list.

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
};
Node* head = nullptr;
void insertAtEnd(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
```

```
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
void reverseList() {
    Node* prev = nullptr;
    Node* curr = head;
    Node* next = nullptr;

    while (curr != nullptr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
}
void print() {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    int value;
    cout << "Enter values to insert (-1 to stop): ";
    while (true) {
        cin >> value;
        if (value == -1) break;
        insertAtEnd(value);
    }
    cout << "Original List: ";
    print();

    reverseList();

    cout << "Reversed List: ";
```

```
    print();

    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 5> cd "c:\Users\ritik\OneDrive\Deskt
Enter values to insert (-1 to stop): 15 2 4 16 48 40 -1
Original List: 15 2 4 16 48 40
Reversed List: 40 48 16 4 2 15
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 5>
```

3. Write a program to merge two sorted linked lists into one sorted linked list.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
Node* mergeSortedLists(Node* head1, Node* head2) {
    if (!head1) return head2;
    if (!head2) return head1;

    Node* result = nullptr;

    if (head1->data <= head2->data) {
        result = head1;
        result->next = mergeSortedLists(head1->next, head2);
    } else {
        result = head2;
        result->next = mergeSortedLists(head1, head2->next);
    }
    return result;
}
void insertAtEnd(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (head == nullptr) {
```

```
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head1 = nullptr;
    Node* head2 = nullptr;
int value;
    cout << "Enter sorted values for first list (-1 to stop): ";
    while (true) {
        cin >> value;
        if (value == -1) break;
        insertAtEnd(head1, value);
    }
    cout << "Enter sorted values for second list (-1 to stop): ";
    while (true) {
        cin >> value;
        if (value == -1) break;
        insertAtEnd(head2, value);
    }
    Node* mergedHead = mergeSortedLists(head1, head2);
    cout << "Merged Sorted List: ";
    displayList(mergedHead);
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 5> cd "c:\Users\ritik\OneDri
Enter sorted values for first list (-1 to stop): 1 4 5 9 8 -1
Enter sorted values for second list (-1 to stop): 2 6 7 9 -1
Merged Sorted List: 1 2 4 5 6 7 9 8 9
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 5>
```

*****

WEEK – 06

LINKED LIST-02

1.Write a program to detect a cycle in a linked list.

```cpp
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
Node* insert(Node* &head, int value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        head =  newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
bool detectCycle(Node* head) {
    if (head == nullptr || head->next == nullptr) return false;
    Node* slow = head;
    Node* fast = head;
    while (fast != nullptr && fast->next != nullptr) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) return true;
    }
    return false;
}
int main() {
    Node* head = nullptr;
    insert(head, 1);
    insert(head, 2);
    insert(head, 3);
    head->next->next->next = head->next; // Creating a cycle
```

```
    if (detectCycle(head)) {
        cout << "Cycle detected" << endl;
    } else {
        cout << "No cycle detected" << endl;
    }
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> c
Cycle detected
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6>
```

2. Write a program to segregate even and odd nodes in a linked list.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = nullptr;
    }};
Node* insert(Node* &head, int value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }}
Node* segregateEvenOdd(Node* head) {
    if (head == nullptr) return nullptr;
    Node* evenStart = nullptr;
    Node* evenEnd = nullptr;
    Node* oddStart = nullptr;
    Node* oddEnd = nullptr;
    Node* curr = head;
    while (curr != nullptr) {
        int value = curr->data;
```

```
        if (value % 2 == 0) {
            if (evenStart == nullptr) {
                evenStart = curr;
                evenEnd = evenStart;
            } else {
                evenEnd->next = curr;
                evenEnd = evenEnd->next;
            }
        } else {
            if (oddStart == nullptr) {
                oddStart = curr;
                oddEnd = oddStart;}
                else {
                oddEnd->next = curr;
                oddEnd = oddEnd->next;}}
        curr = curr->next;
    }
    if (evenStart == nullptr || oddStart == nullptr) return head;
    evenEnd->next = oddStart;
    oddEnd->next = nullptr;
    return evenStart;
}
void print(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = nullptr;
 insert(head, 1);
 insert(head, 2);
 insert(head, 3);
 insert(head, 4);
 insert(head, 5);
 insert(head, 6);
    head = segregateEvenOdd(head);
    print(head);
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> cd '
Node Before:
1 2 3 4 5 6
2 4 6 1 3 5
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6>
```

3. Write a program to find the intersection point of two linked lists.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }};
Node* insert(Node* head, int value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        return newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
        return head;
    }
}
int getLength(Node* head) {
    int length = 0;
    while (head != nullptr) {
        length++;
        head = head->next;
    }
    return length;
}
Node* findIntersection(Node* head1, Node* head2) {
    int len1 = getLength(head1);
    int len2 = getLength(head2);
    if (len1 > len2) {
        int diff = len1 - len2;
        while (diff--) {
```

```
            head1 = head1->next;
        }
    } else if (len2 > len1) {
        int diff = len2 - len1;
        while (diff--) {
            head2 = head2->next;
        }
    }
    while (head1 != nullptr && head2 != nullptr) {
        if (head1 == head2) {
            return head1;
        }
        head1 = head1->next;
        head2 = head2->next;
    }
    return nullptr;
}
int main() {
    Node* head1 = nullptr;
    Node* head2 = nullptr;
    head1 = insert(head1, 1);
    head1 = insert(head1, 2);
    head1 = insert(head1, 3);
    head2 = insert(head2, 9);
    head2 = insert(head2, 10);
    Node* intersection = new Node(15);
    head1->next->next->next = intersection;
    head2->next->next = intersection;
    intersection->next = insert(nullptr, 30);
    Node* intersectingNode = findIntersection(head1, head2);
    if (intersectingNode != nullptr) {
        cout << "Intersection point is at node with value: " <<
intersectingNode->data << endl;
    } else {
        cout << "No intersection point found." << endl;
    }
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> cd "c:\U
Intersection point is at node with value: 15
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6>
```

4. Write a program to remove duplicates from an unsorted linked list.

```
#include <iostream>
#include <unordered_map>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }};
Node* insert(Node* &head, int value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }}
void print(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;}
Node* removeduplicates(Node* &head){
    unordered_map<int ,int>map;
    Node* curr =  head;
    Node* prev = head;
    while(curr!=NULL){
        if(map[curr->data]==0){
            map[curr->data]++;
            prev= curr;
            curr = curr->next; }
        else{
            prev->next = curr->next;
            curr= curr->next;
        }}}
int main() {
Node* head = nullptr;
```

```
 insert(head, 1);
 insert(head, 2);
 insert(head, 5);
 insert(head, 4);
 insert(head, 5);
 insert(head, 1);
    removeduplicates(head);
    print(head);
    return 0;
}
```

```
 PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> cd "c:\U
 Original List: 1 2 5 4 5 1
 1 2 5 4
 PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6>
```

5. Write a program to rotate a linked list clockwise by \(k \) nodes.

```
#include <iostream>
#include <unordered_map>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
Node* insert(Node* &head, int value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }}
void print(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
```

```
    }
    cout << endl;
}
int getLength(Node* head) {
    int length = 0;
    Node* temp = head;
    while (temp != nullptr) {
        length++;
        temp = temp->next;
    }
    return length;
}
Node* rotateClockwise(Node* head, int k) {
    if (head == nullptr || head->next == nullptr || k == 0) {
        return head;
    }
    int length = getLength(head);
    k = k % length;

    if (k == 0) return head;
    Node* temp = head;
    for (int i = 1; i < length - k; i++) {
        temp = temp->next;
    }
    Node* newHead = temp->next;

    Node* last = newHead;
    while (last->next != nullptr) {
        last = last->next;
    }
    last->next = head;
    temp->next = nullptr;

    return newHead;
}
int main() {
Node* head = nullptr;
 insert(head,8);
 insert(head, 2);
 insert(head, 6);
 insert(head, 4);
 insert(head, 5);
 insert(head, 1);
 int k = 3;
cout << "Original list: ";
```

```
print(head);
head = rotateClockwise(head, k);
cout << "After rotating clockwise by " << k << " nodes: ";
print(head);
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> cd "c:\Users\ritik
Original list: 8 2 6 4 5 1
After rotating clockwise by 3 nodes: 4 5 1 8 2 6
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6>
```

6. Write a program to sort a linked list that contains 0s, 1s, and 2s by changing links.

```
#include <iostream>
#include <unordered_map>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
Node* insert(Node* &head, int value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
void print(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
```

```
}
Node* sortlist(Node* head){
    Node* zeroHead = nullptr;
    Node* zeroTail = nullptr;
    Node* oneHead = nullptr;
    Node* oneTail = nullptr;
    Node* twoHead = nullptr;
    Node* twoTail = nullptr;
    Node* current = head;
    while (current != nullptr) {
        int value = current->data;
        if (value == 0) {
            if (zeroHead == nullptr) {
                zeroHead = current;
                zeroTail = zeroHead;
            } else {
                zeroTail->next = current;
                zeroTail = zeroTail->next;
            }
        } else if (value == 1) {
            if (oneHead == nullptr) {
                oneHead = current;
                oneTail = oneHead;
            } else {
                oneTail->next = current;
                oneTail = oneTail->next;
            }
        } else if (value == 2) {
            if (twoHead == nullptr) {
                twoHead = current;
                twoTail = twoHead;
            } else {
                twoTail->next = current;
                twoTail = twoTail->next;
            }
        }
        current = current->next;
    }
    if (zeroTail != nullptr) {
        zeroTail->next = oneHead;
    }
    if (oneTail != nullptr) {
        oneTail->next = twoHead;
    }
    if (twoTail != nullptr) {
```

```
        twoTail->next = nullptr;
    }
    if (zeroHead != nullptr) {
        return zeroHead;
    } else if (oneHead != nullptr) {
        return oneHead;
    } else {
        return twoHead;
    }
}
int main() {
Node* head = nullptr;
 insert(head, 1);
 insert(head, 0);
 insert(head, 1);
 insert(head, 2);
 insert(head, 0);
 insert(head, 1);
    head = sortlist(head);
    print(head);
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> cd "c:\Users\
Originakl list: 1 0 1 2 0 1
0 0 1 1 1 2
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6>
```

7. Write a program to check if a linked list is a palindrome.

```
#include <iostream>
#include <unordered_map>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
Node* insert(Node* &head, int value) {
    Node* newnode = new Node(value);
    if (head == nullptr) {
        head = newnode;
    } else {
```

```
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newnode;
    }
}
void print(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
bool palindrome(Node* head) {
    if (!head) return true;
    Node* slow = head;
    Node* fast = head;
    Node* prev = nullptr;
    while (fast && fast->next) {
        fast = fast->next->next;
        Node* nextNode = slow->next;
        slow->next = prev;
        prev = slow;
        slow = nextNode;
    }
    if (fast) slow = slow->next;
    while (prev && slow) {
        if (prev->data != slow->data) return false;
        prev = prev->next;
        slow = slow->next;
    }
    return true;
}
int main() {
Node* head = nullptr;
 insert(head, 1);
 insert(head, 0);
 insert(head, 2);
 insert(head, 2);
 insert(head, 0);
 insert(head, 1);
 print(head);
    if(palindrome(head)){
```

```
        cout<<"List is palindrome";
    }
    else{
        cout<<"List is not palindrome";
    }
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> cd "c:\User
1 0 2 2 0 1
List is palindrome
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6>
```

8. Write a program to find the nth node from the end of a linked list.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
Node* insert(Node* head, int value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        return newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
        return head;
    }
}
Node* findNthFromEnd(Node* head, int n) {
    Node* first = head;
    Node* second = head;
    for (int i = 0; i < n; i++) {
        if (second == nullptr) {
            return nullptr;
```

```
        }
        second = second->next;
    }
    while (second != nullptr) {
        first = first->next;
        second = second->next;
    }
    return first;
}
void print(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = nullptr;
    head = insert(head, 10);
    head = insert(head, 20);
    head = insert(head, 30);
    head = insert(head, 40);
    head = insert(head, 50);
    cout << "Original list: ";
    print(head);
    int n = 2;
    Node* result = findNthFromEnd(head, n);
    if (result != nullptr) {
        cout << "The " << n << "th node from the end is: " << result->data <<
endl;
    } else {
        cout << "The list has fewer than " << n << " nodes." << endl;
    }
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> cd "c:\Users\ri
Original list: 10 20 30 40 50
The 2th node from the end is: 40
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6>
```

9. Write a program to implement a doubly linked list with operations to add, remove, and display nodes.

```cpp
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node* prev;
    Node(int value) {
        data = value;
        next = nullptr;
        prev = nullptr;
    }};
Node* addNode(Node* head, int value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        return newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
        return head;
    }}
Node* removeNode(Node* head, int value) {
    if (head == nullptr) return head;
    Node* temp = head;
    while (temp != nullptr && temp->data != value) {
        temp = temp->next;}
    if (temp == nullptr) return head;
    if (temp->prev != nullptr) {
        temp->prev->next = temp->next;
    } else {
        head = temp->next;
    }
    if (temp->next != nullptr) {
        temp->next->prev = temp->prev;
    }
    delete temp;
    return head;}
void print(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next; }
```

```
    cout << endl;}
int main() {
    Node* head = nullptr;
    head = addNode(head, 10);
    head = addNode(head, 20);
    head = addNode(head, 30);
    head = addNode(head, 40);
    cout << "List: ";
    print(head);
    head = removeNode(head, 20);
    cout << "After removing 20: ";
    print(head);
    head = removeNode(head, 40);
    cout << "After removing 40: ";
    print(head);
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> cd "c
List: 10 20 30 40
After removing 20: 10 30 40
After removing 40: 10 30
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6>
```

10. Write a program to implement a linked list where a node consists of a student's data (like name, age, and score).

```
#include <iostream>
#include <string>
using namespace std;
struct Student {
    string name;
    int age;
    float score;
    Student* next;
    Student(string n, int a, float s) {
        name = n;
        age = a;
        score = s;
        next = nullptr;
    }};
Student* addStudent(Student* head, string name, int age, float score) {
    Student* newStudent = new Student(name, age, score);
    if (head == nullptr) {
        return newStudent;
```

```
    } else {
        Student* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newStudent;
        return head;
    }}
void displayStudents(Student* head) {
    Student* temp = head;
    while (temp != nullptr) {
        cout << "Name: " << temp->name << ", Age: " << temp->age << ", Score: "
<< temp->score << endl;
        temp = temp->next;
    }}
int main() {
    Student* head = nullptr;
    head = addStudent(head, "Alice", 20, 85.5);
    head = addStudent(head, "Bob", 22, 90.0);
    head = addStudent(head, "Charlie", 21, 88.7);
    cout << "Student List:" << endl;
    displayStudents(head);
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> cd "c:\Users\rit
Student List:
Name: Alice, Age: 20, Score: 85.5
Name: Bob, Age: 22, Score: 90
Name: Charlie, Age: 21, Score: 88.7
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6>
```

11. Create a linked list to manage inventory items for a small store. Each node should hold information about the product, such as the product ID, name, quantity, and price. Provide functions to add new products, delete products, and restock products.

```
#include <iostream>
#include <string>
using namespace std;
struct Product {
    int productID;
    string name;
    int quantity;
    float price;
    Product* next;
```

```
    Product(int id, string n, int q, float p) {
        productID = id;
        name = n;
        quantity = q;
        price = p;
        next = nullptr;
    }};
Product* addProduct(Product* head, int productID, string name, int quantity,
float price) {
    Product* newProduct = new Product(productID, name, quantity, price);
    if (head == nullptr) {
        return newProduct;
    } else {
        Product* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;}
        temp->next = newProduct;
        return head;
    }}
Product* deleteProduct(Product* head, int productID) {
    if (head == nullptr) return head;
    Product* temp = head;
    Product* prev = nullptr;
    if (temp != nullptr && temp->productID == productID) {
        head = temp->next;
        delete temp;
        return head;
    }
    while (temp != nullptr && temp->productID != productID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == nullptr) return head;
    prev->next = temp->next;
    delete temp;
    return head;}
void restockProduct(Product* head, int productID, int additionalQuantity) {
    Product* temp = head;
    while (temp != nullptr && temp->productID != productID) {
        temp = temp->next;
    }
    if (temp != nullptr) {
        temp->quantity += additionalQuantity;
        cout << "Product " << temp->name << " restocked. New quantity: " <<
temp->quantity << endl;
```

```
    } else {
        cout << "Product with ID " << productID << " not found." << endl;
    }}
void displayInventory(Product* head) {
    Product* temp = head;
    if (temp == nullptr) {
        cout << "Inventory is empty." << endl;
    } else {
        while (temp != nullptr) {
            cout << "Product ID: " << temp->productID << ", Name: " << temp-
>name
                 << ", Quantity: " << temp->quantity << ", Price: $" << temp-
>price << endl;
            temp = temp->next;
        }}}
int main() {
    Product* inventory = nullptr;
    inventory = addProduct(inventory, 101, "Apple", 50, 0.99);
    inventory = addProduct(inventory, 102, "Banana", 100, 0.59);
    inventory = addProduct(inventory, 103, "Orange", 80, 1.29);
    cout << "Inventory after adding products:" << endl;
    displayInventory(inventory);
    restockProduct(inventory, 102, 20);
    cout << "Inventory after restocking Banana:" << endl;
    displayInventory(inventory);
    inventory = deleteProduct(inventory, 101);
    cout << "Inventory after deleting Apple:" << endl;
    displayInventory(inventory);
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> cd "c:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6\" ;
Inventory after adding products:
Product ID: 101, Name: Apple, Quantity: 50, Price: $0.99
Product ID: 102, Name: Banana, Quantity: 100, Price: $0.59
Product ID: 103, Name: Orange, Quantity: 80, Price: $1.29
Product Banana restocked. New quantity: 120
Inventory after restocking Banana:
Product ID: 101, Name: Apple, Quantity: 50, Price: $0.99
Product ID: 102, Name: Banana, Quantity: 120, Price: $0.59
Product ID: 103, Name: Orange, Quantity: 80, Price: $1.29
Inventory after deleting Apple:
Product ID: 102, Name: Banana, Quantity: 120, Price: $0.59
Product ID: 103, Name: Orange, Quantity: 80, Price: $1.29
```

12. Write a program to manage an emergency room queue using a linked list. Each node represents a patient with attributes like patient ID, name, and emergency level. Patients should be sorted by emergency level. Provide functions to add a patient, treat the next

patient, and display the queue.

```
#include <iostream>
#include <string>
using namespace std;
struct Patient {
    int patientID;
    string name;
    int emergencyLevel;
    Patient* next;
    Patient(int id, string n, int level) {
        patientID = id;
        name = n;
        emergencyLevel = level;
        next = nullptr;
    }
};
Patient* addPatient(Patient* head, int patientID, string name, int
emergencyLevel) {
    Patient* newPatient = new Patient(patientID, name, emergencyLevel);
    if (head == nullptr || head->emergencyLevel < emergencyLevel) {
        newPatient->next = head;
        return newPatient;
    }
    Patient* temp = head;
    while (temp->next != nullptr && temp->next->emergencyLevel >=
emergencyLevel) {
        temp = temp->next;
    }
    newPatient->next = temp->next;
    temp->next = newPatient;
    return head;
}
Patient* treatNextPatient(Patient* head) {
    if (head == nullptr) {
        cout << "No patients in the queue." << endl;
        return head;
    }
    Patient* temp = head;
    cout << "Treating patient: " << temp->name << " (ID: " << temp->patientID <<
", Emergency Level: " << temp->emergencyLevel << ")" << endl;
    head = head->next;
    delete temp;
    return head;
}
```

```cpp
void displayQueue(Patient* head) {
    if (head == nullptr) {
        cout << "No patients in the queue." << endl;
        return;
    }
    Patient* temp = head;
    while (temp != nullptr) {
        cout << "Patient ID: " << temp->patientID << ", Name: " << temp->name <<
", Emergency Level: " << temp->emergencyLevel << endl;
        temp = temp->next;
    }
}
int main() {
    Patient* head = nullptr;
    head = addPatient(head, 101, "Alice", 3);
    head = addPatient(head, 102, "Bob", 5);
    head = addPatient(head, 103, "Charlie", 2);
    head = addPatient(head, 104, "Diana", 4);
    cout << "Emergency Room Queue:" << endl;
    displayQueue(head);
    head = treatNextPatient(head);
    cout << "Queue after treating the next patient:" << endl;
    displayQueue(head);
    return 0;
}
```

```
PS C:\Users\ritik\OneDrive\Desktop\lab\dsa\lab 6> cd "c:\Users\ritik\OneD
Emergency Room Queue:
Patient ID: 102, Name: Bob, Emergency Level: 5
Patient ID: 104, Name: Diana, Emergency Level: 4
Patient ID: 101, Name: Alice, Emergency Level: 3
Patient ID: 103, Name: Charlie, Emergency Level: 2
Treating patient: Bob (ID: 102, Emergency Level: 5)
Queue after treating the next patient:
Patient ID: 104, Name: Diana, Emergency Level: 4
Patient ID: 101, Name: Alice, Emergency Level: 3
Patient ID: 103, Name: Charlie, Emergency Level: 2
```

******

WEEK – 07

Linked List – 07

1. Write a program to merge two circular linked lists into a single circular linked list. Display the merged list after the operation.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int value){
        data = value;
        next = NULL;
    }
};
void insertNode(Node* &head, int value) {
    Node* newNode = new Node(value);
    if (head == NULL) {
        head = newNode;
        newNode->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
}
void print(Node* head) {
    if (!head) return;
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}
Node* mergeCircularLists(Node* head1, Node* head2) {
    if (!head1) return head2;
    if (!head2) return head1;
```

```
    Node* temp1 = head1;
    while (temp1->next != head1) {
        temp1 = temp1->next;
    }
    Node* temp2 = head2;
    while (temp2->next != head2) {
        temp2 = temp2->next;
    }
    temp1->next = head2;
    temp2->next = head1;
    return head1;  // Returning the merged list starting from head1
}
int main() {
    Node* head1 = nullptr;
    Node* head2 = nullptr;
    insertNode(head1, 1);
    insertNode(head1, 2);
    insertNode(head1, 3);
    insertNode(head2, 4);
    insertNode(head2, 5);
    insertNode(head2, 6);
    cout << "First Circular Linked List: ";
 print(head1);
    cout << "Second Circular Linked List: ";
 print(head2);
    Node* mergedHead = mergeCircularLists(head1, head2);
    cout << "Merged Circular Linked List: ";
 print(mergedHead);
    return 0;
}
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
    Node(int value){
        data = value;
        next = NULL;
    }
};
void insertNode(Node* &head, int value) {
    Node* newNode = new Node(value);
    if (head == NULL) {
        head = newNode;
```

```
        newNode->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
}
void print(Node* head) {
    if (!head) return;
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}
Node* mergeCircularLists(Node* head1, Node* head2) {
    if (!head1) return head2;
    if (!head2) return head1;
    Node* temp1 = head1;
    while (temp1->next != head1) {
        temp1 = temp1->next;
    }
    Node* temp2 = head2;
    while (temp2->next != head2) {
        temp2 = temp2->next;
    }
    temp1->next = head2;
    temp2->next = head1;
    return head1;  // Returning the merged list starting from head1
}
int main() {
    Node* head1 = nullptr;
    Node* head2 = nullptr;
    insertNode(head1, 1);
    insertNode(head1, 2);
    insertNode(head1, 3);
    insertNode(head2, 4);
    insertNode(head2, 5);
    insertNode(head2, 6);
    cout << "First Circular Linked List: ";
 print(head1);
```

```
    cout << "Second Circular Linked List: ";
 print(head2);
    Node* mergedHead = mergeCircularLists(head1, head2);
    cout << "Merged Circular Linked List: ";
 print(mergedHead);
    return 0;
}
```

```
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7> cd "c:\Users\Ritik
First Circular Linked List: 1 2 3
Second Circular Linked List: 4 5 6
Merged Circular Linked List: 1 2 3 4 5 6
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7>
```

2. Write a program to split a circular linked list into two halves. If the number of nodes is odd, the extra node should go into the first list.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int value){
        data = value;
        next = NULL;
    }
};
void insertNode(Node* &head, int value) {
    Node* newNode = new Node(value);
    if (head == NULL) {
        head = newNode;
        newNode->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next; }
        temp->next = newNode;
        newNode->next = head;}}
void print(Node* head) {
    if (!head) return;
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
```

```
}
void splitList(Node* head, Node*& head1, Node*& head2) {
    if (!head || head->next == head) {
        head1 = head;
        head2 = nullptr;
        return;
    }
    Node* slow = head;
    Node* fast = head;

    while (fast->next != head && fast->next->next != head) {
        slow = slow->next;
        fast = fast->next->next;
    }
    if (fast->next->next == head) {
        fast = fast->next;
    }
    head1 = head;
    head2 = slow->next;
    slow->next = head1;
    fast->next = head2;
}
int main() {
   Node* head = nullptr;
    Node* head1 = nullptr;
    Node* head2 = nullptr;
    insertNode(head, 1);
    insertNode(head, 2);
    insertNode(head, 3);
    insertNode(head, 4);
    insertNode(head, 5);
    cout << "Original Circular Linked List: ";
    print(head);
    splitList(head, head1, head2);
    cout << "First Half: ";
    print(head1);
    cout << "Second Half: ";
    print(head2);
    return 0;}
```

```
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7> cd "c:\Users\
Original Circular Linked List: 1 2 3 4 5
First Half: 1 2 3
Second Half: 4 5
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7>
```

3. Develop a program to find and return the middle element of a circular linked list.

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
    Node(int value){
        data = value;
        next = NULL;
    }
};
void insertNode(Node* &head, int value) {
    Node* newNode = new Node(value);
    if (head == NULL) {
        head = newNode;
        newNode->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
}
void print(Node* head) {
    if (!head) return;
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}

Node* findMiddle(Node* head){
    Node* slow = head;
    Node* fast = head;

    while(fast->next!=head && fast->next->next!=head){
        fast = fast->next->next;
        slow = slow->next;
    }
```

```
    return slow;
}

int main() {
   Node* head = nullptr;
    insertNode(head, 1);
    insertNode(head, 2);
    insertNode(head, 3);
    insertNode(head, 4);
    insertNode(head, 5);
    cout << "Circular Linked List: ";
    print(head);
    Node* middle = findMiddle(head);
    if (middle) {
        cout << "Middle Element: " << middle->data << endl;
    } else {
        cout << "The list is empty." << endl;
    }
}
```

```
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7> cd "c:\Users\Ri
Circular Linked List: 1 2 3 4 5
Middle Element: 3
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7>
```

4. Write a program to concatenate two circular linked lists into one circular linked list.

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

void insertNode(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
        newNode->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
```

```
        }
        temp->next = newNode;
        newNode->next = head;
    }
}

void displayList(Node* head) {
    if (!head) return;
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}

void concatenateLists(Node*& head1, Node*& head2) {
    if (!head1) {
        head1 = head2;
        return;
    }
    if (!head2) return;

    Node* temp1 = head1;
    while (temp1->next != head1) {
        temp1 = temp1->next;
    }

    Node* temp2 = head2;
    while (temp2->next != head2) {
        temp2 = temp2->next;
    }

    temp1->next = head2;
    temp2->next = head1;
}

int main() {
    Node* head1 = nullptr;
    Node* head2 = nullptr;
    insertNode(head1, 1);
    insertNode(head1, 2);
    insertNode(head1, 3);
    insertNode(head2, 4);
    insertNode(head2, 5);
```

```
    insertNode(head2, 6);
    cout << "First Circular Linked List: ";
    displayList(head1);
    cout << "Second Circular Linked List: ";
    displayList(head2);
    concatenateLists(head1, head2);
    cout << "Concatenated Circular Linked List: ";
    displayList(head1);
    return 0;
}
```

```
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7> cd
First Circular Linked List: 1 2 3
Second Circular Linked List: 4 5 6
Concatenated Circular Linked List: 1 2 3 4 5 6
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7>
```

5. Write a program to check whether a given circular linked list is sorted in ascending order.

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};
void insertNode(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
        newNode->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
}
bool isSorted(Node* head) {
    if (!head || head->next == head) return true;

    Node* temp = head;
```

```
    do {
        if (temp->data > temp->next->data) {
            return false;
        }
        temp = temp->next;
    } while (temp->next != head);

    return true;
}
void displayList(Node* head) {
    if (!head) return;
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}
int main() {
    Node* head = nullptr;
    insertNode(head, 1);
    insertNode(head, 2);
    insertNode(head, 3);
    insertNode(head, 4);
    insertNode(head, 5);
    cout << "Circular Linked List: ";
    displayList(head);

    if (isSorted(head)) {
        cout << "The circular linked list is sorted in ascending order." <<
endl;
    } else {
        cout << "The circular linked list is not sorted in ascending order." <<
endl;
    }

    return 0;
}
```

```
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7
Circular Linked List: 1 2 3 4 5
The circular linked list is sorted in ascending order.
```

6. Write a program to check whether the elements of a circular linked list form a palindrome.

```
#include <iostream>
using namespace std;
```

```
struct Node {
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

void insertNode(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
        newNode->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
}

void displayList(Node* head) {
    if (!head) return;
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}

Node* reverseList(Node* head) {
    Node* prev = nullptr;
    Node* current = head;
    Node* next = nullptr;

    do {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    } while (current != head);
```

```
    head->next = prev;
    return prev;
}

bool isPalindrome(Node* head) {
    if (!head || head->next == head) return true;

    Node* slow = head;
    Node* fast = head;

    while (fast->next != head && fast->next->next != head) {
        slow = slow->next;
        fast = fast->next->next;
    }

    Node* secondHalf = slow->next;
    slow->next = head;
    secondHalf = reverseList(secondHalf);

    Node* firstHalf = head;
    Node* secondHalfStart = secondHalf;

    bool palindrome = true;
    do {
        if (firstHalf->data != secondHalf->data) {
            palindrome = false;
            break;
        }
        firstHalf = firstHalf->next;
        secondHalf = secondHalf->next;
    } while (secondHalf != secondHalfStart);

    reverseList(secondHalfStart);
    slow->next = secondHalfStart;

    return palindrome;
}

int main() {
    Node* head = nullptr;
    insertNode(head, 1);
    insertNode(head, 2);
    insertNode(head, 3);
    insertNode(head, 2);
    insertNode(head, 1);
```

```
    cout << "Circular Linked List: ";
    displayList(head);
    if (isPalindrome(head)) {
        cout << "The circular linked list is a palindrome." << endl;
    } else {
        cout << "The circular linked list is not a palindrome." << endl;
    }
    return 0;
}
```

```
Circular Linked List: 1 2 3 2 1
The circular linked list is not a palindrome.
```

7. Write a program to create a doubly linked list and perform the following operations:

a) Insert at the beginning

b) Insert at the end

c) Insert at a specific position.

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
    Node* prev;

    Node(int value) : data(value), next(nullptr), prev(nullptr) {}
};

void insertAtBeginning(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}

void insertAtEnd(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
    } else {
```

```
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

void insertAtPosition(Node*& head, int value, int position) {
    if (position <= 1) {
        insertAtBeginning(head, value);
        return;
    }

    Node* newNode = new Node(value);
    Node* temp = head;
    int count = 1;

    while (temp && count < position - 1) {
        temp = temp->next;
        count++;
    }

    if (!temp) {
        cout << "Position out of bounds. Inserting at the end." << endl;
        insertAtEnd(head, value);
    } else {
        newNode->next = temp->next;
        newNode->prev = temp;
        if (temp->next) {
            temp->next->prev = newNode;
        }
        temp->next = newNode;
    }
}

void displayList(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
```

```
int main() {
    Node* head = nullptr;

    insertAtBeginning(head, 10);
    insertAtEnd(head, 20);
    insertAtEnd(head, 30);
    insertAtPosition(head, 15, 2);
    cout << "Doubly Linked List: ";
    displayList(head);
  return 0;
}
```

```
Doubly Linked List: 10 15 20 30
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7>
```

8. Write a program to delete nodes from a doubly linked list:

a) Delete the first node

b) Delete the last node

c) Delete a node at a specific position

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
    Node* prev;

    Node(int value) : data(value), next(nullptr), prev(nullptr) {}
};

void insertAtEnd(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}
```

```
void deleteFirstNode(Node*& head) {
    if (!head) return;

    Node* temp = head;
    head = head->next;
    if (head) {
        head->prev = nullptr;
    }
    delete temp;
}

void deleteLastNode(Node*& head) {
    if (!head) return;

    if (!head->next) {
        delete head;
        head = nullptr;
        return;
    }

    Node* temp = head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->prev->next = nullptr;
    delete temp;
}

void deleteAtPosition(Node*& head, int position) {
    if (!head) return;

    if (position <= 1) {
        deleteFirstNode(head);
        return;
    }

    Node* temp = head;
    int count = 1;
    while (temp && count < position) {
        temp = temp->next;
        count++;
    }
    if (!temp) {
        cout << "Position out of bounds." << endl;
```

```
        return;
    }
    if (temp->next) {
        temp->next->prev = temp->prev; }
    if (temp->prev) {
        temp->prev->next = temp->next; }
    delete temp;}
void displayList(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = nullptr;
    insertAtEnd(head, 10);
    insertAtEnd(head, 20);
    insertAtEnd(head, 30);
    insertAtEnd(head, 40);
    insertAtEnd(head, 50);
    cout << "Doubly Linked List: ";
    displayList(head);
    deleteFirstNode(head);
    cout << "After deleting the first node: ";
    displayList(head);
    deleteLastNode(head);
    cout << "After deleting the last node: ";
    displayList(head);
    deleteAtPosition(head, 2);
    cout << "After deleting the node at position 2: ";
    displayList(head);
    return 0;
}
```

```
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7> cd "c:\Users\Riti
Doubly Linked List: 10 20 30 40 50
After deleting the first node: 20 30 40 50
After deleting the last node: 20 30 40
After deleting the node at position 2: 20 40
```

9. Write a program to traverse a doubly linked list in both forward and reverse directions and print the data in each node.

```
#include <iostream>
using namespace std;
```

```
struct Node {
    int data;
    Node* next;
    Node* prev;

    Node(int value) : data(value), next(nullptr), prev(nullptr) {}
};

void insertAtEnd(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

void displayForward(Node* head) {
    Node* temp = head;
    cout << "Traversing Forward: ";
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

void displayReverse(Node* head) {
    if (!head) return;

    Node* temp = head;
    while (temp->next) {
        temp = temp->next;
    }

    cout << "Traversing Reverse: ";
    while (temp) {
        cout << temp->data << " ";
        temp = temp->prev;
```

```
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    insertAtEnd(head, 10);
    insertAtEnd(head, 20);
    insertAtEnd(head, 30);
    insertAtEnd(head, 40);
    insertAtEnd(head, 50);
    // Traverse the list in both forward and reverse directions
    displayForward(head);
    displayReverse(head);
    return 0;
}
```

```
Traversing Forward: 10 20 30 40 50
Traversing Reverse: 50 40 30 20 10
```

10. Develop a program to calculate and return the length (number of nodes) of a doubly linked list.

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
    Node* prev;

    Node(int value) : data(value), next(nullptr), prev(nullptr) {}
};
void insertAtEnd(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}
int getLength(Node* head) {
```

```
    int length = 0;
    Node* temp = head;
    while (temp) {
        length++;
        temp = temp->next;
    }
    return length;
}
int main() {
    Node* head = nullptr;
    insertAtEnd(head, 10);
    insertAtEnd(head, 20);
    insertAtEnd(head, 30);
    insertAtEnd(head, 40);
    insertAtEnd(head, 50);
    cout << "Length of the doubly linked list: " << getLength(head) << endl;

    return 0;
}
```

```
PS C:\Users\Ritik gupta\OneDrive\Desktop\lab\dsa\lab 7> cd "c:\Us
Length of the doubly linked list: 5
```

*****

WEEK – 08

STACK

1. Implement a stack using an array. Write functions for push, pop, and display operations.

```cpp
#include <iostream>
using namespace std;
class stack{
    int arr[1000];
    int size;
    public:
    stack(){
        size=0;
    }
    void push(int d){
        if(size>1000){return;}
        arr[++size] = d;
    }
    void pop(){
        size--;
    }
    int top(){
        return arr[size];
    }
    bool empty(){
        if(size==0)
        return true;
        else
        return false;
    }
};
int main(){
    stack st;
    st.push(2);
    st.push(4);
    st.push(6);
    cout<<st.top()<<endl;
    st.pop();
    cout<<st.top()<<endl;
    cout<<st.empty();
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik g
6
4
0
```

2. Implement a stack using a linked list. Write functions to push, pop, and display stack elements.

```
#include <iostream>
using namespace std;
class Node{
    public:
    int data;
    Node* next;

    Node(int d){
        data = d;
        next =NULL;
    }
};
class stack{
    public:
    Node* head = NULL;

    void push(int d){
        Node* temp = head;
        Node* curr = new Node(d);
        if(temp == NULL){head = curr;}
        else{
            head = curr;
            head->next = temp;
        }
    }
    void pop(){
        head = head->next;
    }
    int top(){
        return head->data;
    }
    bool empty(){
        if(head==NULL){return true;}
        return false;
    }
};
int main(){
    stack st;
    st.push(2);
    st.push(4);
    st.push(6);
     st.push(4);
    cout<<st.top()<<endl;
```

```
    st.pop();
    cout<<st.top()<<endl;
    cout<<st.empty();
}
```

```
4
6
0
```

3. Write a program to reverse a string using a stack. Demonstrate how each character is pushed and then popped to achieve the reversal.

```
#include <iostream>
using namespace std;
#include <stack>

int main(){
    string s;
    cout<<"Enter string: "<<endl;
    cin>>s;

    stack <char>st;
    for(char c : s ){
        st.push(c);
    }

    string ans;

   while(!st.empty()){
    ans.push_back(st.top());
    st.pop();   }

    cout<<"Reversed string: "<<ans;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desktop\lab\dsa\lab 8\" ; if ($?) { 
Enter string:
Ritik
Reversed string: kitiR
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 8>
```

4. Evaluate a postfix expression using a stack. For example, evaluate the expression 6 2 + 5 * 8 4 / -.

```
#include <iostream>
#include <stack>
#include <sstream>
#include <string>
using namespace std;
int evaluatePostfix(const string& expression) {
    stack<int> s;
    stringstream ss(expression);
```

```
    string token;
    while (ss >> token) {
        if (isdigit(token[0])) {
            s.push(stoi(token));
        }
        else {
            int operand2 = s.top(); s.pop();
            int operand1 = s.top(); s.pop();
            int result = 0;

            switch (token[0]) {
                case '+': result = operand1 + operand2; break;
                case '-': result = operand1 - operand2; break;
                case '*': result = operand1 * operand2; break;
                case '/': result = operand1 / operand2; break;
                default: cout << "Invalid operator encountered!" << endl; return
-1;
            }
            s.push(result);
        }
    }
    return s.top();
}
int main() {
    string expression = "6 2 + 5 * 8 4 / -";
    cout << "The result of the expression is: " << evaluatePostfix(expression)
<< endl;
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desktop\lab\dsa\lab 8\" ; if ($?
The result of the expression is: 38
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 8>
```

5. Convert an infix expression to postfix notation using a stack (e.g., converting (A + B) * (C-D)).

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}
string infixToPostfix(const string& expression) {
    stack<char> s;
```

```
    string postfix = "";

    for (char token : expression) {
        // If the character is an operand, add it to output
        if (isalnum(token)) {
            postfix += token;
        }
        // If the character is '(', push it to stack
        else if (token == '(') {
            s.push(token);
        }
        // If the character is ')', pop and output from the stack until '(' is
found
        else if (token == ')') {
            while (!s.empty() && s.top() != '(') {
                postfix += s.top();
                s.pop();
            }
            s.pop();  // Remove '(' from the stack
        }
        // An operator is encountered
        else {
            while (!s.empty() && precedence(s.top()) >= precedence(token)) {
                postfix += s.top();
                s.pop();
            }
            s.push(token);
        }
    }
    while (!s.empty()) {
        postfix += s.top();
        s.pop();
    }

    return postfix;
}
int main() {
    string expression = "(A+B)*(C-D)";
    cout << "Postfix expression: " << infixToPostfix(expression) << endl;
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desktop\lab\dsa\lab 8\" ; if ($?) { g++ 5.c++ -o 5
Postfix expression: AB+CD-*
```

6. Check for balanced parentheses in an expression using a stack. This includes (), {}, and [] pairs.

```
#include <iostream>
```

```cpp
#include <stack>
#include <string>
using namespace std;
bool isMatchingPair(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '{' && close == '}') ||
           (open == '[' && close == ']');
}
bool isBalanced(const string& expression) {
    stack<char> s;
    for (char ch : expression) {
        if (ch == '(' || ch == '{' || ch == '[') {
            s.push(ch);
        }
        else if (ch == ')' || ch == '}' || ch == ']') {
            if (s.empty() || !isMatchingPair(s.top(), ch)) {
                return false;
            }
            s.pop();
        }
    }
    return s.empty();
}
int main() {
    string expression = "{[()]}";
    if (isBalanced(expression)) {
        cout << "The expression is balanced." << endl;
    } else {
        cout << "The expression is not balanced." << endl;
    }
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desktc
The expression is balanced.
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 8>
```

7. Implement a function to sort a stack. The function should use only one additional stack for assistance.

```cpp
#include <iostream>
#include <stack>
using namespace std;
void sortStack(stack<int>& inputStack) {
    stack<int> auxStack;
    while (!inputStack.empty()) {
        int temp = inputStack.top();
```

```
        inputStack.pop();
        while (!auxStack.empty() && auxStack.top() > temp) {
            inputStack.push(auxStack.top());
            auxStack.pop();
        }
        auxStack.push(temp);
    }
    while (!auxStack.empty()) {
        inputStack.push(auxStack.top());
        auxStack.pop();
    }
}
void printStack(stack<int> s) {
    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }
    cout << endl;
}
int main() {
    stack<int> s;
    s.push(34);
    s.push(3);
    s.push(31);
    s.push(98);
    s.push(92);
    s.push(23);

    cout << "Original Stack: ";
    printStack(s);

    sortStack(s);

    cout << "Sorted Stack: ";
    printStack(s);

    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desktop\lab\dsa\lab 8\" ; if ($?) { g++ 7.c++ -o 7 } ; if ($?)
Original Stack: 23 92 98 31 3 34
Sorted Stack: 3 23 31 34 92 98
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 8>
```

8. Write a program to find the minimum element in a stack at any point, with push and pop operations.

```
#include <iostream>
#include <stack>
using namespace std;
```

```cpp
class MinStack {
    stack<int> mainStack;
    stack<int> minStack;
public:
    void push(int x) {
        mainStack.push(x);
        if (minStack.empty() || x <= minStack.top()) {
            minStack.push(x);
        }
    }
    void pop() {
        if (mainStack.top() == minStack.top()) {
            minStack.pop();
        }
        mainStack.pop();
    }
    int getMin() {
        return minStack.top();
    }
    int top() {
        return mainStack.top();
    }
};
int main() {
    MinStack s;
    s.push(5);
    s.push(7);
    s.push(3);
    s.push(7);
    s.push(3);
    cout << "Minimum element: " << s.getMin() << endl;  // Output: 3
    s.pop();
    cout << "Minimum element: " << s.getMin() << endl;  // Output: 3
    s.pop();
    cout << "Minimum element: " << s.getMin() << endl;  // Output: 3
    s.pop();
    cout << "Minimum element: " << s.getMin() << endl;  // Output: 5

    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desktop\lab\dsa\lab 8\" ; if ($?) { g++ 8.c++ -o 8 } ; if ($?) { 
Minimum element: 3
Minimum element: 3
Minimum element: 3
Minimum element: 5
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 8>
```

9. Write a program to find the maximum element in a stack at any point, with push and pop operations.

```cpp
#include <iostream>
#include <stack>
#include <stdexcept>
using namespace std;
class MaxStack {
    stack<int> mainStack;
    stack<int> maxStack;
public:
    void push(int x) {
        mainStack.push(x);
        if (maxStack.empty() || x >= maxStack.top()) {
            maxStack.push(x);
        }
    }
    void pop() {
        if (mainStack.empty()) {
            throw runtime_error("Stack is empty");
        }
        if (mainStack.top() == maxStack.top()) {
            maxStack.pop();
        }
        mainStack.pop();
    }
    int getMax(){
        if (maxStack.empty()) {
            throw runtime_error("Stack is empty");
        }
        return maxStack.top();
    }
    int top() {
        if (mainStack.empty()) {
            throw runtime_error("Stack is empty");
        }
        return mainStack.top();
    }
};
int main() {
    MaxStack s;
    s.push(5);
    s.push(3);
```

```
    s.push(7);
    s.push(3);

    cout << "Maximum element: " << s.getMax() << endl;  // Output: 7
    s.pop();
    cout << "Maximum element: " << s.getMax() << endl;  // Output: 7
    s.pop();
    cout << "Maximum element: " << s.getMax() << endl;  // Output: 5
    s.pop();
    cout << "Maximum element: " << s.getMax() << endl;  // Output: 5

    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desktop\lab\dsa\lab 8\" ; if ($?) { g++
Maximum element: 7
Maximum element: 7
Maximum element: 5
Maximum element: 5
```

10. Convert a given postfix expression to infix notation using a stack.

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}

// Function to convert postfix to infix
string postfixToInfix(const string& postfix) {
    stack<string> s;
    for (char token : postfix) {
        // If the token is an operand, push it as a string to the stack
        if (isalnum(token)) {
            s.push(string(1, token));
        }
        // If the token is an operator
        else if (isOperator(token)) {
            // Pop two operands from the stack
            string operand2 = s.top(); s.pop();
            string operand1 = s.top(); s.pop();

            // Form the infix expression and push it back to the stack
            string infixExpression = "(" + operand1 + token + operand2 + ")";
            s.push(infixExpression);
        }
```

```
    }
    // The remaining element in the stack is the full infix expression
    return s.top();
}

int main() {
    string postfix = "AB+C*D-";
    cout << "Infix expression: " << postfixToInfix(postfix) << endl;
    return 0;
}
```

```
Infix expression: (((A+B)*C)-D)
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 8>
```

QUEUE

1. Write a program to implement a queue using an array with basic operations: enqueue, dequeue, and display. Include checks for queue overflow and underflow.

```
#include <iostream>
using namespace std;

class Queue {
    int front, rear, size;
    int *queue;
    int capacity;

public:
    Queue(int capacity) {
        this->capacity = capacity;
        front = rear = -1;
        queue = new int[capacity];
    }

    void enqueue(int value) {
        if (rear == capacity - 1) {
            cout << "Queue Overflow\n";
```

```
            return;
        }
        if (front == -1) front = 0;
        queue[++rear] = value;
    }

    void dequeue() {
        if (front == -1 || front > rear) {
            cout << "Queue Underflow\n";
            return;
        }
        cout << "Dequeued: " << queue[front++] << endl;
    }

    void display() {
        if (front == -1 || front > rear) {
            cout << "Queue is empty\n";
            return;
        }
        for (int i = front; i <= rear; i++) {
            cout << queue[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    Queue q(5);
    q.enqueue(10);
    q.enqueue(20);
    q.display();
    q.dequeue();
    q.display();
    return 0;
}

```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desktop\lab\dsa
10 20
Dequeued: 10
20
```

2. Write a program to implement a queue using a linked list with basic operations: enqueue, dequeue, and display.

```
#include <iostream>
```

```cpp
using namespace std;

struct Node {
    int data;
    Node* next;
};

class Queue {
    Node *front, *rear;

public:
    Queue() {
        front = rear = nullptr;
    }
    void enqueue(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;
        if (rear == nullptr) {
            front = rear = newNode;
            return;
        }
        rear->next = newNode;
        rear = newNode;
    }

    void dequeue() {
        if (front == nullptr) {
            cout << "Queue Underflow\n";
            return;
        }
        Node* temp = front;
        front = front->next;
        if (front == nullptr) rear = nullptr;
        cout << "Dequeued: " << temp->data << endl;
        delete temp;
    }

    void display() {
        if (front == nullptr) {
            cout << "Queue is empty\n";
            return;
        }
        Node* temp = front;
        while (temp != nullptr) {
```

```
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    Queue q;
    q.enqueue(10);
    q.enqueue(20);
    q.display();
    q.dequeue();
    q.display();
    return 0;
}
```

```
10 20
Dequeued: 10
20
```

3. Write a program to count the number of elements in a queue implemented using either an array or a linked list.

```
#include <iostream>
using namespace std;

class Queue {
    int front, rear;
    int size, capacity;
    int *queue;

public:
    Queue(int capacity) {
        this->capacity = capacity;
        front = rear = -1;
        size = 0;
        queue = new int[capacity];
    }

    void enqueue(int value) {
        if (rear == capacity - 1) {
            cout << "Queue Overflow\n";
            return;
```

```
        }
        if (front == -1) front = 0;
        queue[++rear] = value;
        size++;
    }

    void dequeue() {
        if (front == -1 || front > rear) {
            cout << "Queue Underflow\n";
            return;
        }
        front++;
        size--;
    }

    int count() {
        return size;
    }
};

int main() {
    Queue q(5);
    q.enqueue(10);
    q.enqueue(20);
    cout << "Count: " << q.count() << endl;
    q.dequeue();
    cout << "Count: " << q.count() << endl;
    return 0;
}
```

```
Count: 2
Count: 1
```

4. Write a program that implements a queue and includes a peek operation to display the front element of the queue without removing it.

```
#include <iostream>
using namespace std;

class Queue {
    int front, rear, size, capacity;
    int *queue;

public:
    Queue(int capacity) {
```

```
        this->capacity = capacity;
        front = rear = -1;
        queue = new int[capacity];
    }

    void enqueue(int value) {
        if (rear == capacity - 1) {
            cout << "Queue Overflow\n";
            return;
        }
        if (front == -1) front = 0;
        queue[++rear] = value;
    }

    void dequeue() {
        if (front == -1 || front > rear) {
            cout << "Queue Underflow\n";
            return;
        }
        cout << "Dequeued: " << queue[front++] << endl;
    }

    void peek() {
        if (front == -1 || front > rear) {
            cout << "Queue is empty\n";
            return;
        }
        cout << "Front Element: " << queue[front] << endl;
    }
};

int main() {
    Queue q(5);
    q.enqueue(10);
    q.enqueue(20);
    q.peek();
    q.dequeue();
    q.peek();
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desktop\
Front Element: 10
Dequeued: 10
Front Element: 20
```

5. Write a program to reverse the elements of a queue using only stack operations.

```cpp
#include <iostream>
#include <queue>
#include <stack>
using namespace std;

void reverseQueue(queue<int> &q) {
    stack<int> s;
    while (!q.empty()) {
        s.push(q.front());
        q.pop();
    }
    while (!s.empty()) {
        q.push(s.top());
        s.pop();
    }
}

int main() {
    queue<int> q;
    q.push(10);
    q.push(20);
    q.push(30);

    cout << "Original Queue: ";
    queue<int> temp = q;
    while (!temp.empty()) {
        cout << temp.front() << " ";
        temp.pop();
    }
    cout << endl;

    reverseQueue(q);

    cout << "Reversed Queue: ";
    while (!q.empty()) {
        cout << q.front() << " ";
```

```
        q.pop();
    }
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta
Original Queue: 10 20 30
Reversed Queue: 30 20 10
```

6. Write a program to implement a circular queue using an array. Implement enqueue,dequeue, and display operations, and handle circular indexing.

```
#include <iostream>
using namespace std;

class CircularQueue {
    int front, rear, size;
    int *queue;

public:
    CircularQueue(int capacity) {
        size = capacity;
        front = rear = -1;
        queue = new int[capacity];
    }

    void enqueue(int value) {
        if ((rear + 1) % size == front) {
            cout << "Queue Overflow\n";
            return;
        }
        if (front == -1) front = 0;
        rear = (rear + 1) % size;
        queue[rear] = value;
    }

    void dequeue() {
        if (front == -1) {
            cout << "Queue Underflow\n";
            return;
        }
        cout << "Dequeued: " << queue[front] << endl;
        if (front == rear) front = rear = -1;
        else front = (front + 1) % size;
```

```
    }

    void display() {
        if (front == -1) {
            cout << "Queue is empty\n";
            return;
        }
        int i = front;
        while (true) {
            cout << queue[i] << " ";
            if (i == rear) break;
            i = (i + 1) % size;
        }
        cout << endl;
    }
};

int main() {
    CircularQueue cq(5);
    cq.enqueue(10);
    cq.enqueue(20);
    cq.display();
    cq.dequeue();
    cq.display();
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desktop\lab\dsa\lab 8.2
10 20
Dequeued: 10
20
```

7. Write a program that finds the minimum element in a queue without altering the queue's content. Display the minimum element without dequeuing it.

```
#include <iostream>
#include <queue>

using namespace std;

int findMin(queue<int> q) {
    int minValue = INT_MAX;
    while (!q.empty()) {
        if (q.front() < minValue) minValue = q.front();
        q.pop();
```

```
    }
    return minValue;
}

int main() {
    queue<int> q;
    q.push(30);
    q.push(10);
    q.push(20);

    cout << "Minimum Element: " << findMin(q) << endl;
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\
Minimum Element: 10
```

8. Write a program to merge two queues into a third queue. The resulting queue should contain elements from both queues in their original order.

```
#include <iostream>
#include <queue>
using namespace std;

queue<int> mergeQueues(queue<int> q1, queue<int> q2) {
    queue<int> merged;
    while (!q1.empty()) {
        merged.push(q1.front());
        q1.pop();
    }
    while (!q2.empty()) {
        merged.push(q2.front());
        q2.pop();
    }
    return merged;
}

int main() {
    queue<int> q1, q2;
    q1.push(10);
    q1.push(20);
    q2.push(30);
    q2.push(40);
```

```
    queue<int> merged = mergeQueues(q1, q2);

    cout << "Merged Queue: ";
    while (!merged.empty()) {
        cout << merged.front() << " ";
        merged.pop();
    }
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desk
Merged Queue: 10 20 30 40
```

9. Write a program to implement a queue using two stacks. Implement enqueue and dequeue operations.

```
#include <iostream>
#include <stack>
using namespace std;

class Queue {
    stack<int> s1, s2;

public:
    void enqueue(int value) {
        s1.push(value);
    }

    void dequeue() {
        if (s1.empty() && s2.empty()) {
            cout << "Queue Underflow\n";
            return;
        }
        if (s2.empty()) {
            while (!s1.empty()) {
                s2.push(s1.top());
                s1.pop();
            }
        }
        cout << "Dequeued: " << s2.top() << endl;
        s2.pop();
    }
};

int main() {
```

```
    Queue q;
    q.enqueue(10);
    q.enqueue(20);
    q.dequeue();
    q.dequeue();
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\Desktop\
Dequeued: 10
Dequeued: 20
```

10. Write a program to find and display the sum of all elements in a queue without modifying the queue's content.

```
#include <iostream>
#include <queue>
using namespace std;

int sumQueue(queue<int> q) {
    int sum = 0;
    while (!q.empty()) {
        sum += q.front();
        q.pop();
    }
    return sum;
}

int main() {
    queue<int> q;
    q.push(10);
    q.push(20);
    q.push(30);

    cout << "Sum of Queue: " << sumQueue(q) << endl;
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gupta\D
Sum of Queue: 60
```

11. Write a program that uses a queue to check if a string is a palindrome. Enqueue each character, then dequeue to verify the order.

```cpp
#include <iostream>
#include <queue>
using namespace std;

bool isPalindrome(string s) {
    queue<char> q;
    for (char c : s) q.push(c);

    for (int i = 0; i < s.size(); i++) {
        if (q.front() != s[s.size() - 1 - i]) return false;
        q.pop();
    }
    return true;
}

int main() {
    string s = "madam";
    if (isPalindrome(s)) cout << "Palindrome\n";
    else cout << "Not a Palindrome\n";
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\U
Palindrome
```

12. Write a program that takes a queue and an integer k as input and reverses the first k elements of the queue, leaving the rest in the same order.

```cpp
#include <iostream>
#include <queue>
#include <stack>
using namespace std;

void reverseFirstK(queue<int> &q, int k) {
    stack<int> s;
    for (int i = 0; i < k; i++) {
        s.push(q.front());
        q.pop();
    }
    while (!s.empty()) {
        q.push(s.top());
        s.pop();
    }
    for (int i = 0; i < q.size() - k; i++) {
```

```
        q.push(q.front());
        q.pop();
    }
}

int main() {
    queue<int> q;
    q.push(10);
    q.push(20);
    q.push(30);
    q.push(40);
    q.push(50);

    int k = 3;
    reverseFirstK(q, k);

    cout << "Modified Queue: ";
    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa> cd "c:\Users\Ritik gu
Modified Queue: 30 20 10 40 50
```

****

WEEK – 09

BINARY TREE

1. Write a C program to implement a simple tree structure and perform insertion & Deletion of nodes.

```
#include <iostream>

using namespace std;

class Node {
    public:
    int data;
    Node* left;
    Node* right;
    Node(int d) {
    data = d;
    left = NULL;
    right = NULL;
    }
};
Node* insertNode(Node* root, int data) {
    if (root == nullptr) return new Node(data);
    if (data < root->data)
        root->left = insertNode(root->left, data);
    else
        root->right = insertNode(root->right, data);
    return root;
}
Node* findMin(Node* root) {
    while (root->left != nullptr) root = root->left;
    return root;
}
Node* deleteNode(Node* root, int data) {
    if (root == nullptr) return root;
    if (data < root->data)
        root->left = deleteNode(root->left, data);
    else if (data > root->data)
        root->right = deleteNode(root->right, data);
    else {
        if (root->left == nullptr) {
            Node* temp = root->right;
```

```
            delete root;
            return temp;
        } else if (root->right == nullptr) {
            Node* temp = root->left;
            delete root;
            return temp;
        }
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
void inorderTraversal(Node* root) {
    if(root) {
        inorderTraversal(root->left);
        cout << root->data << " ";
        inorderTraversal(root->right);
    }  }

int main() {
    Node* root = nullptr;
    root = insertNode(root, 50);
    root = insertNode(root, 30);
    root = insertNode(root, 20);
    root = insertNode(root, 40);
    root = insertNode(root, 70);
    root = insertNode(root, 60);
    root = insertNode(root, 80);
    inorderTraversal(root);
    cout<<endl;
    root = deleteNode(root, 50);
    root = deleteNode(root,70);
    inorderTraversal(root);
    return 0;
}
```

```
20 30 40 50 60 70 80
20 30 40 60 80
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9>
```

2. Write a C program to find the height of a tree.

```
#include <iostream>
using namespace std;
```

```
class Node {
    public:
    int data;
    Node* left;
    Node* right;
    Node(int d) {
    data = d;
    left = NULL;
    right = NULL;
    }
};
int findHeight(Node* root) {
    if(root==NULL)return 0;
    int left = findHeight(root->left);
    int right = findHeight(root->right);
    int ans = max(left,right) +1;
    return ans;
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    cout << findHeight(root);
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9> cd "
3
```

3. Write a C program to implement a function that checks whether a given tree is symmetric.

```
#include <iostream>
using namespace std;
class TreeNode {
public:
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int val) {
```

```
        data = val;
        left = nullptr;
        right = nullptr;
    }
};
    bool isSymmetric(TreeNode* leftTree, TreeNode* rightTree) {


        if(leftTree == NULL && rightTree == NULL)return true;
        if(leftTree == NULL || rightTree == NULL)return false;
        return (leftTree->data == rightTree->data) && isSymmetric(leftTree-
>left,rightTree->right) && isSymmetric(leftTree->right , rightTree->left);
    }
int main() {
     TreeNode* root = new TreeNode(1);
     root->left = new TreeNode(2);
     root->right = new TreeNode(2);
     root->left->left = new TreeNode(3);
     root->left->right = new TreeNode(4);
     root->right->left = new TreeNode(4);
     root->right->left = new TreeNode(3);

    cout << (isSymmetric(root->left,root->right) ? "Symmetric" : "Not
Symmetric");
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9> cd "c:
Not Symmetric
```

4. Write a C program to perform a pre-order/in-order/post-order/level-order traversal of a binary tree.

```
#include <iostream>
#include <queue>
using namespace std;

class Node {
    public:
    int data;
    Node* left;
    Node* right;
    Node(int d) {
    data = d;
    left = NULL;
```

```
    right = NULL;
    }
};
Node* insertNode(Node* root, int data) {
    if (root == nullptr) return new Node(data);
    if (data < root->data)
        root->left = insertNode(root->left, data);
    else
        root->right = insertNode(root->right, data);
    return root;
}
    void preorder(Node* node) {
        if(node==NULL)return;
        cout<<node->data<<" ";
        preorder(node->left);
        preorder(node->right);
    }

    void inorder(Node* node) {
        if (!node) return;
        inorder(node->left);
        cout << node->data << " ";
        inorder(node->right);
    }

    void postorder(Node* node) {
        if (!node) return;
        postorder(node->left);
        postorder(node->right);
        cout << node->data << " ";
    }

    void levelOrder(Node* node) {
         if(node==NULL)return;
        queue<Node*>q;
        q.push(node);
        while(!q.empty()){
            Node* temp = q.front();
            q.pop();
            cout<<temp->data<<" ";
            if(temp->left) q.push(temp->left);
            if(temp->right)q.push(temp->right);
        }

    }
```

```
int main() {
    Node* root;
    root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->left = new Node(6);
    root->right->right = new Node(7);

    cout << "Pre-order: ";
    preorder(root);
    cout << "\nIn-order: ";
    inorder(root);
    cout << "\nPost-order: ";
    postorder(root);
    cout << "\nLevel-order: ";
    levelOrder(root);
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9> cd "c:\Users\Ri
Pre-order: 1 2 4 5 3 6 7
In-order: 4 2 5 1 6 3 7
Post-order: 4 5 2 6 7 3 1
Level-order: 1 2 3 4 5 6 7
```

5. Write a C program to count the total number of internal(non-leaf) and leaf Node in a binary tree.

```
#include <iostream>
using namespace std;

class Node {
    public:
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};
```

```
int countLeaves(Node* node) {
    if(node==NULL)return 0;
    if(node->left ==NULL && node->right==NULL)return 1;
    return countLeaves(node->left) + countLeaves(node->right);
}

int countNonLeaves(Node* node) {
    if (!node || (!node->left && !node->right)) return 0;
    return 1 + countNonLeaves(node->left) + countNonLeaves(node->right);
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->left = new Node(6);

    cout << "Leaf nodes: " << countLeaves(root) << endl;
    cout << "Non-leaf nodes: " << countNonLeaves(root) << endl;
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9> cd "c:\Users\R
Leaf nodes: 3
Non-leaf nodes: 3
```

6. Write a C program to check if two binary trees are identical.

```
#include <iostream>
using namespace std;

class Node {
    public:
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
```

```
};
bool areIdentical(Node* tree1, Node* tree2) {
    if (!tree1 && !tree2) return true;
    if (!tree1 || !tree2) return false;
    return (tree1->data == tree2->data) &&
           areIdentical(tree1->left, tree2->left) &&
           areIdentical(tree1->right, tree2->right);
}
int main() {
    Node* root1 = new Node(1);
    root1->left = new Node(2);
    root1->right = new Node(3);

    Node* root2 = new Node(1);
    root2->left = new Node(2);
    root2->right = new Node(3);

    cout << (areIdentical(root1, root2) ? "Identical" : "Not Identical");
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9> cd "c:\Users\Ri
Identical
```

7. Write a C program to implement mirror conversion of a binary tree.

```
#include <iostream>
using namespace std;

class Node {
    public:
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};
void mirrorTree(Node* root) {
    if (!root) return;
    swap(root->left, root->right);
    mirrorTree(root->left);
```

```
    mirrorTree(root->right);
}
void inorderTraversal(Node* root) {
    if (root) {
        inorderTraversal(root->left);
        cout << root->data << " ";
        inorderTraversal(root->right);
    }
}
int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);

    cout << "Original Tree: ";
    inorderTraversal(root);
    cout << "\n";
    mirrorTree(root);
    cout << "Mirrored Tree: ";
    inorderTraversal(root);
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9> cd "c:\Users\Ritik gupta\
Original Tree: 4 2 5 1 3
Mirrored Tree: 3 1 5 2 4
```

8. Write a C program to create a binary search tree and insert/delete nodes into it.

```
#include <iostream>
using namespace std;

class Node {
    public:
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
```

```
};

Node* insertNode(Node* root, int data) {
    if (!root) return new Node(data);
    if (data < root->data)
        root->left = insertNode(root->left, data);
    else if (data > root->data)
        root->right = insertNode(root->right, data);
    return root;
}

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

Node* deleteNode(Node* root, int data) {
    if (!root) return root;
    if (data < root->data)
        root->left = deleteNode(root->left, data);
    else if (data > root->data)
        root->right = deleteNode(root->right, data);
    else {
        if (!root->left) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (!root->right) {
            Node* temp = root->left;
            delete root;
            return temp;
        }
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

void inorderTraversal(Node* root) {
    if (root) {
        inorderTraversal(root->left);
        cout << root->data << " ";
        inorderTraversal(root->right);
    }
```

```
}

int main() {
    Node* root = nullptr;
    root = insertNode(root, 50);
    root = insertNode(root, 30);
    root = insertNode(root, 70);
    root = insertNode(root, 20);
    root = insertNode(root, 40);
    root = insertNode(root, 60);
    root = insertNode(root, 80);

    cout << "BST Inorder: ";
    inorderTraversal(root);
    cout << "\n";

    root = deleteNode(root, 50);
    root = deleteNode(root, 20);

    cout << "BST after deletion: ";
    inorderTraversal(root);
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9> cd "c:\Users\Ritik
BST Inorder: 20 30 40 50 60 70 80
BST after deletion: 30 40 60 70 80
```

9. Write a C program to search for a given value in a binary search tree.

```
#include <iostream>
using namespace std;

class Node {
    public:
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};
```

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

int main() {
    Node* root = nullptr;
    root = new Node(50);
    root->left = new Node(30);
    root->right = new Node(70);
    root->left->left = new Node(20);
    root->left->right = new Node(40);

    int key = 40;
    Node* result = search(root, key);
    if (result)
        cout << "Found: " << result->data;
    else
        cout << "Not Found";
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9> cd "c:\Users\Riti
Found: 40
```

10. Write a C program to find the minimum and maximum values in a binary search tree.

```
#include <iostream>
using namespace std;

class Node {
    public:
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};
```

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

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

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

int main() {
    Node* root = nullptr;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);

    Node* minNode = findMin(root);
    Node* maxNode = findMax(root);

    if (minNode) cout << "Minimum value: " << minNode->data << endl;
    if (maxNode) cout << "Maximum value: " << maxNode->data << endl;

    return 0;
}
```

```
Minimum value: 20
Maximum value: 80
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9>
```

11. Write a C program to check if a binary tree is a binary search tree (BST).

```cpp
#include <iostream>
using namespace std;

class Node {
    public:
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};

bool isBSTUtil(Node* root, Node* minNode, Node* maxNode) {
    if (!root) return true;
    if ((minNode && root->data <= minNode->data) ||
        (maxNode && root->data >= maxNode->data))
        return false;
    return isBSTUtil(root->left, minNode, root) &&
           isBSTUtil(root->right, root, maxNode);
}

bool isBST(Node* root) {
    return isBSTUtil(root, nullptr, nullptr);
}

int main() {
    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(15);
    root->left->left = new Node(2);
    root->left->right = new Node(7);

    cout << (isBST(root) ? "The tree is a BST" : "The tree is not a BST");
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9> cd "c:\Users\Riti
The tree is a BST
```

12. Write a C program to print all elements of a BST within a given range.

```cpp
#include <iostream>
using namespace std;

class Node {
    public:
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};

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

void printRange(Node* root, int low, int high) {
    if (!root) return;
    if (root->data > low) printRange(root->left, low, high);
    if (root->data >= low && root->data <= high) cout << root->data << " ";
    if (root->data < high) printRange(root->right, low, high);
}

int main() {
    Node* root = nullptr;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);

    int low = 35, high = 65;
    cout << "Elements in range [" << low << ", " << high << "]: ";
    printRange(root, low, high);
```

```
    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9> cd "c:\Users\Ritik
Elements in range [35, 65]: 40 50 60
```

13. Write a C program to find the k-th smallest element in a BST.

```
#include <iostream>
using namespace std;

class Node {
    public:
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};

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

int kSmallest(Node* root, int& k) {
    if (!root) return -1;
    int left = kSmallest(root->left, k);
    if (left != -1) return left;
    k--;
    if (k == 0) return root->data;
    return kSmallest(root->right, k);
}

int main() {
    Node* root = nullptr;
```

```
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);

    int k = 3;
    int result = kSmallest(root, k);
    if (result != -1)
        cout << "The " << 3 << "-rd smallest element is: " << result;
    else
        cout << "Less than " << k << " elements in the BST";

    return 0;
}
```

```
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 9> cd "c:\Users\
The 3-rd smallest element is: 40
```

*****

AVL TREE

1. Write a C program to implement an AVL tree and perform insertion/Deletion/Search operations.

```
#include <iostream>
using namespace std;

struct Node {
    int key;
    Node* left;
    Node* right;
    int height;
};

int height(Node* n) {
    return n == nullptr ? 0 : n->height;
}

Node* createNode(int key) {
    Node* node = new Node();
    node->key = key;
    node->left = nullptr;
    node->right = nullptr;
    node->height = 1;
    return node;
}

int getBalanceFactor(Node* n) {
    return n == nullptr ? 0 : height(n->left) - height(n->right);
}

Node* rightRotate(Node* y) {
    Node* x = y->left;
    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;
}

Node* leftRotate(Node* x) {
    Node* y = x->right;
    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;
}

Node* insert(Node* node, int key) {
    if (node == nullptr)
        return createNode(key);
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else
        return node;
    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalanceFactor(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;
}

Node* minValueNode(Node* node) {
    Node* current = node;
    while (current->left != nullptr)
        current = current->left;
    return current;
}

Node* deleteNode(Node* root, int key) {
    if (root == nullptr)
        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 == nullptr) || (root->right == nullptr)) {
            Node* temp = root->left ? root->left : root->right;
            if (temp == nullptr) {
                temp = root;
                root = nullptr;
            } else
                *root = *temp;
            delete temp;
        } else {
            Node* temp = minValueNode(root->right);
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
        }
    }
    if (root == nullptr)
        return root;
    root->height = 1 + max(height(root->left), height(root->right));
    int balance = getBalanceFactor(root);
    if (balance > 1 && getBalanceFactor(root->left) >= 0)
        return rightRotate(root);
    if (balance > 1 && getBalanceFactor(root->left) < 0) {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    if (balance < -1 && getBalanceFactor(root->right) <= 0)
        return leftRotate(root);
    if (balance < -1 && getBalanceFactor(root->right) > 0) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }
    return root;
}

bool search(Node* root, int key) {
    if (root == nullptr)
        return false;
    if (key == root->key)
        return true;
    else if (key < root->key)
        return search(root->left, key);
    else
        return search(root->right, key);
```

```
}

void inOrder(Node* root) {
    if (root != nullptr) {
        inOrder(root->left);
        cout << root->key << " ";
        inOrder(root->right);
    }
}

int main() {
    Node* root = nullptr;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
    root = insert(root, 25);
    cout << "In-order traversal of the AVL tree: ";
    inOrder(root);
    cout << endl;
    root = deleteNode(root, 30);
    cout << "After deletion of 30, In-order traversal: ";
    inOrder(root);
    cout << endl;
    int searchKey = 25;
    cout << "Search " << searchKey << ": " << (search(root, searchKey) ? "Found"
: "Not Found") << endl;
    return 0;
}
```

```
In-order traversal of the AVL tree: 10 20 25 30 40 50
After deletion of 30, In-order traversal: 10 20 25 40 50
Search 25: Found
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 10>
```

2. Write a C program to find minimum. maximum and Kth minimum element in an AVL tree.

```
#include <iostream>
using namespace std;

struct Node {
    int key;
```

```
    Node* left;
    Node* right;
    int height;
};

int height(Node* n) {
    return n == nullptr ? 0 : n->height;
}

Node* createNode(int key) {
    Node* node = new Node();
    node->key = key;
    node->left = nullptr;
    node->right = nullptr;
    node->height = 1;
    return node;
}

int getBalanceFactor(Node* n) {
    return n == nullptr ? 0 : height(n->left) - height(n->right);
}

Node* rightRotate(Node* y) {
    Node* x = y->left;
    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;
}

Node* leftRotate(Node* x) {
    Node* y = x->right;
    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;
}

Node* insert(Node* node, int key) {
    if (node == nullptr)
        return createNode(key);
```

```
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else
        return node;
    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalanceFactor(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;
}

void inOrder(Node* root) {
    if (root != nullptr) {
        inOrder(root->left);
        cout << root->key << " ";
        inOrder(root->right);
    }
}

Node* minValueNode(Node* node) {
    Node* current = node;
    while (current->left != nullptr)
        current = current->left;
    return current;
}

Node* maxValueNode(Node* node) {
    Node* current = node;
    while (current->right != nullptr)
        current = current->right;
    return current;
}
```

```cpp
int kthSmallestUtil(Node* root, int& k) {
    if (root == nullptr)
        return -1;
    int left = kthSmallestUtil(root->left, k);
    if (left != -1)
        return left;
    k--;
    if (k == 0)
        return root->key;
    return kthSmallestUtil(root->right, k);
}

int kthSmallest(Node* root, int k) {
    return kthSmallestUtil(root, k);
}

int main() {
    Node* root = nullptr;
    root = insert(root, 20);
    root = insert(root, 10);
    root = insert(root, 30);
    root = insert(root, 5);
    root = insert(root, 15);
    root = insert(root, 25);
    root = insert(root, 35);
    cout << "In-order traversal of the AVL tree: ";
    inOrder(root);
    cout << endl;
    Node* minNode = minValueNode(root);
    Node* maxNode = maxValueNode(root);
    cout << "Minimum value: " << (minNode ? minNode->key : -1) << endl;
    cout << "Maximum value: " << (maxNode ? maxNode->key : -1) << endl;
    int k = 3;
    cout << k << "rd smallest element: " << kthSmallest(root, k) << endl;
    return 0;
}
```

```
In-order traversal of the AVL tree: 5 10 15 20 25 30 35
Minimum value: 5
Maximum value: 35
3rd smallest element: 15
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 10>
```

3. Write a program to implement Kruskal's algorithm for finding the Minimum Spanning Tree (MST) of a graph and display the cost of the MST.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Edge {
    int src, dest, weight;
};
bool compareEdge(const Edge& a, const Edge& b) {
    return a.weight < b.weight;
}
struct Subset {
    int parent, rank;
};
int find(Subset subsets[], int i) {
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);
    return subsets[i].parent;
}
void Union(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++;
    }
}
void kruskalMST(vector<Edge>& edges, int V) {
    sort(edges.begin(), edges.end(), compareEdge);
    vector<Edge> result;
    Subset* subsets = new Subset[V];
    for (int v = 0; v < V; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }
    int mstWeight = 0;
    for (Edge& edge : edges) {
        int x = find(subsets, edge.src);
        int y = find(subsets, edge.dest);
```

```
        if (x != y) {
            result.push_back(edge);
            mstWeight += edge.weight;
            Union(subsets, x, y);
        }
    }
    cout << "Edges in the Minimum Spanning Tree:\n";
    for (Edge& edge : result) {
        cout << edge.src << " -- " << edge.dest << " == " << edge.weight <<
endl;
    }
    cout << "Total cost of the MST: " << mstWeight << endl;
    delete[] subsets;
}
int main() {
    int V = 4;
    vector<Edge> edges = {
        {0, 1, 10}, {0, 2, 6}, {0, 3, 5}, {1, 3, 15}, {2, 3, 4}
    };
    kruskalMST(edges, V);
    return 0;
}
```

```
Edges in the Minimum Spanning Tree:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Total cost of the MST: 19
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 10>
```

4. Write a program to implement Prim's algorithm for finding the Minimum Spanning Tree (MST) of a graphand display the cost of the MST.

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

void primMST(vector<vector<int>>& graph, int V) {
    vector<int> key(V, INT_MAX);
    vector<bool> inMST(V, false);
    vector<int> parent(V, -1);
    key[0] = 0;
    int mstCost = 0;
    for (int count = 0; count < V - 1; ++count) {
        int minKey = INT_MAX, u;
```

```
        for (int v = 0; v < V; ++v) {
            if (!inMST[v] && key[v] < minKey) {
                minKey = key[v];
                u = v;
            }
        }
        inMST[u] = true;
        for (int v = 0; v < V; ++v) {
            if (graph[u][v] && !inMST[v] && graph[u][v] < key[v]) {
                key[v] = graph[u][v];
                parent[v] = u;
            }
        }
    }
    cout << "Edges in the Minimum Spanning Tree:\n";
    for (int i = 1; i < V; ++i) {
        cout << parent[i] << " -- " << i << " == " << graph[i][parent[i]] <<
endl;
        mstCost += graph[i][parent[i]];
    }
    cout << "Total cost of the MST: " << mstCost << endl;
}
int main() {
    int V = 5;
    vector<vector<int>> graph = {
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0}
    };
    primMST(graph, V);
    return 0;
}
```

```
Edges in the Minimum Spanning Tree:
0 -- 1 == 2
1 -- 2 == 3
0 -- 3 == 6
1 -- 4 == 5
Total cost of the MST: 16
```

5. Write a C program to perform Depth First Search (DFS) on a graph.

```
#include <iostream>
```

```cpp
#include <vector>
#include <climits>

using namespace std;

void dfs(int node, vector<vector<int>>& graph, vector<bool>& visited,
vector<int>& dist) {
    visited[node] = true;

    for (int neighbor : graph[node]) {
        if (!visited[neighbor]) {
            dist[neighbor] = dist[node] + 1;
            dfs(neighbor, graph, visited, dist);
        }
    }
}
int main() {
    int V = 5, E = 6;
    vector<vector<int>> graph(V);
    vector<bool> visited(V, false);
    vector<int> dist(V, INT_MAX);
    int source = 0;
    graph[0].push_back(1);
    graph[1].push_back(0);
    graph[0].push_back(2);
    graph[2].push_back(0);
    graph[1].push_back(3);
    graph[3].push_back(1);
    graph[2].push_back(3);
    graph[3].push_back(2);
    graph[3].push_back(4);
    graph[4].push_back(3);
    dist[source] = 0;
    dfs(source, graph, visited, dist);

    for (int i = 0; i < V; i++) {
        if (dist[i] == INT_MAX) {
            cout << i << ": Not reachable" << endl;
        } else {
            cout << i << ": " << dist[i] << endl;
        }
    }

    return 0;
}
```

```
0: 0
1: 1
2: 3
3: 2
4: 3
```

6. Write a C program to perform Breadth First Search (BFS) on a graph.

```cpp
#include <iostream>
#include <list>
#include <queue>
#include <vector>
using namespace std;
class Graph {
public:
    int V;
    list<int> *adj;
    Graph(int V);
    void addEdge(int v, int w);
    void BFS(int start);
};
Graph::Graph(int V) {
    this->V = V;
    adj = new list<int>[V];
}
void Graph::addEdge(int v, int w) {
    adj[v].push_back(w);  // Add w to v's list
    adj[w].push_back(v);  // Add v to w's list (undirected graph)
}
void Graph::BFS(int start) {
    vector<bool> visited(V, false);  // Mark all vertices as not visited
    queue<int> q;  // Create a queue for BFS
    visited[start] = true;
    q.push(start);

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";
        for (auto adjNode : adj[node]) {
            if (!visited[adjNode]) {
                visited[adjNode] = true;
                q.push(adjNode);
            }
        }
```

```
    }
}

int main() {
    Graph g(6);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 5);

    cout << "Breadth First Search starting from node 0:" << endl;
    g.BFS(0);

    return 0;
}
```

```
Breadth First Search starting from node 0:
0 1 2 3 4 5
PS C:\Users\Ritik gupta\Desktop\lab\dsa\lab 10>
```

7. Write a C program to find the shortest path from a source node to all other nodes in an unweighted graph using BFS.

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;
void findShortestPath(vector<vector<int>> &adj, int source, int nodes) {
    vector<int> distance(nodes, INT_MAX);
    queue<int> q;
    distance[source] = 0;
    q.push(source);
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        for (int neighbor : adj[node]) {
            if (distance[neighbor] == INT_MAX) {
                distance[neighbor] = distance[node] + 1;
                q.push(neighbor);
            }
        }
    }
```

```
    cout << "Shortest distances from source node " << source << ":\n";
    for (int i = 0; i < nodes; i++) {
        cout << "Node " << i << ": " << distance[i] << "\n";
    }
}
int main() {
    int nodes = 6;
    vector<vector<int>> adj = {
        {1, 2},
        {0, 3, 4},
        {0, 4},
        {1, 5},
        {1, 2, 5},
        {3, 4}
    };
    int source = 0;
    findShortestPath(adj, source, nodes);
    return 0;
}
```

```
Shortest distances from source node 0:
Node 0: 0
Node 1: 1
Node 2: 1
Node 3: 2
Node 4: 2
Node 5: 3
```

8. Write a C program to implement Dijkstra's algorithm for finding the shortest path from a source to all vertices in a weighted graph.

```
#include <stdio.h>
#include <limits.h>
#define MAX 100
#define INF INT_MAX
void dijkstra(int graph[MAX][MAX], int n, int source) {
    int distance[MAX], visited[MAX] = {0};
    for (int i = 0; i < n; i++) {
        distance[i] = INF;
    }
    distance[source] = 0;
    for (int count = 0; count < n - 1; count++) {
        int minDistance = INF, u = -1;
        for (int i = 0; i < n; i++) {
            if (!visited[i] && distance[i] < minDistance) {
                minDistance = distance[i];
```

```
                u = i; }}
        if (u == -1) break;
        visited[u] = 1;
        for (int v = 0; v < n; v++) {
            if (!visited[v] && graph[u][v] && distance[u] != INF &&
                distance[u] + graph[u][v] < distance[v]) {
                distance[v] = distance[u] + graph[u][v];
            }
        }
    }
    printf("Shortest distances from source node %d:\n", source);
    for (int i = 0; i < n; i++) {
        if (distance[i] == INF) {
            printf("Node %d: Unreachable\n", i);
        } else {
            printf("Node %d: %d\n", i, distance[i]);
        }
    }
}
int main() {
    int n = 5;
    int graph[MAX][MAX] = {
        {0, 10, 0, 30, 100},
        {10, 0, 50, 0, 0},
        {0, 50, 0, 20, 10},
        {30, 0, 20, 0, 60},
        {100, 0, 10, 60, 0}
    };
    int source = 0;  // Starting node
    dijkstra(graph, n, source);
    return 0;
}
```

```
Shortest distances from source node 0:
Node 0: 0
Node 1: 10
Node 2: 50
Node 3: 30
Node 4: 60
```

******