

# **ABES Engineering College, Ghaziabad**

(Affiliated to Dr.A.P.J AKTU, Lucknow)

Department of Computer Science



## **Lab Manual**

### **Session 2022-23 (Odd Semester)**

**Student Name : Anubhav Singhal(2100320120024)**

**Branch-Section : CS-A**

**Subject Name : Data Structures Using 'C'**

**Subject Code : KCS 351**

**Semester : B.Tech. CS III (A,B,C)**

**Faculty Name(s) : MR. GAURAV VATS  
MR. DHANESHWAR KUMAR  
MR. CHANDRAHAS MISHRA**

## **Tools & Software**

### **Recommended Systems/Software Requirements:**

Turbo C/C++

DEV C++

Code Block

Any Online C/C++ Compiler

NO.	LAB No.	Name of The Experiment	Date
1	Lab1	Program for traversing array elements.	
2		Program to insert the given elements into an array.	
3		Program for insertion in the sorted array	
4		Program for delete the given elements into an array.	
5	Lab 2	Program for Missing number in an array	
6		Program to find which element is repeated in the array and which is not	
7		Program for reversal of an array.	
8		Program for merging two sorted arrays	
9	Lab 3	Program for Matrix Addition	
10		Program for Matrix Subtraction	
11		Program for Matrix Multiplication	
12	Lab 4	Program for Matrix Transpose	
13		Program for finding Matrix Determinant	
14		Program for Matrix transposition without second matrix	
15	Lab 5	program for Linear Search	
16		Program for Binary search	
17		Program for Index sequential Search	
18	Lab 6	Program for Hash Table Implementation for Basic Hash Function (Without collisions)	
19		Program for Hash Table Implementation for Collision Resoulution using Linear Probing	



40	Lab 12	Program for Stack Primitive Operations	
41		Program for Decimal to Any Base Conversion	
42		Program to check the validity of Parenthesized Arithmetic Expression using Stack	
43	Lab 13	Program to check the validity of Bracketed Arithmetic Expression using Stack	
44		Program to check if the given number is a palindrome using stacks	
45		Program to Reverse the given String using Stack	
46	Lab 14	Program for Postfix Evaluation	
47		Program for Prefix Evaluation	
48		Program for Infix to Postfix Conversion	
49		Program for Infix to Prefix Conversion	
50	Lab 15	Program for implementation of 2 stacks using a single Array	
51		Program for Finding Minimum in the Stack	
52		Program for Sorting of stack	
53		Program for implementation of Multiple stack in one Array	
54	Lab 16	Program of Array Implementation of Linear Queue	
55		Program of Array Implementation of Circular Queue	
56		Program for Array Implementation of Double Ended Queue	
57	Lab 17	Program for Array Implementation of Priority Queue (Ascending Array)	
58		Program for Array Implementation of Priority Queue (Descending Array)	
59		Program for Stack implementation using Queue	
60		Program for Queue implementation using Stack	

61	Lab 18	Program for Linear Linked List Primitive operations	
62		Program for creation of Linked List header file and test of basic functions through that	
63		Program for finding count of Nodes in Linked List	
64		Program for concatenation of Linear Linked List	
65		Program to implement Linear search.	
66	Lab 19	Program to insert an item at any given position in the linked List	
67		Program for Creation of Copy of the Linked list	
68		Program for counting nodes containing even and odd information.	
69		Program for Creation of Ascending Order Linear Linked List	
70		Program for Merging two sorted Linked List/unsoted link list	
71		Program for finding difference of two linked list (consider lists as sets)	
72	Lab 20	Program for Finding the Middle element of a singly linked list in one pass	
73		Program to perform Binary Search on the Linked List	
74		Program for Reversing the Linear Linked List	
75		Program to print Linked List contents in reverse order	
76		Program for Pair wise swap of elements in linked list	
77		Program to find kth node from the last in a single link list	
78		Program for Sorting the Linear Linked List	

79	Lab 21	Program to Detect if there is any cycle in the linked list, starting point of cycle, length of cycle	
80		Program for Delete duplicate nodes in the Linked List	
81		Program for Linked List Implementation of Priority Queue	
82		Program to arrange the consonants and vowel nodes of the linked list in such a way that all the vowel nodes come before the consonants while maintaining the order of their arrival	
83		Program for Deletion of all occurrences of x from Linked List	
84		Program to Delete kth node from end of a linked list in a single scan and O(n) time	
85	Lab 22	Program to find out the addition of two given linked lists 125+85 = 210 1->2->5 8->5	
86		Program to find out the subtraction of two given linked lists	
87		Program for Polynomial Addition using Linked List	
88		Program for Polynomial Multiplication using Linked List	
89	Lab 23	Program for Circular Linked List Primitive Operations	
90		Program for concatenation of Circular Linked List	
91		Program for implementation of Josephus Problem	
92	Lab 24	Program for Doubly linked list Primitive operations	
93		Program for Circular Doubly Linked List Primitive Operations	
94		Program for Linked List Implementation of Stacks	
95		Program for Linked List Implementation of Queue	
96		Program for Linked List implementation of Double Ended Queue	

97	Lab 25	Program for Pre-Order, In-Order, Post-Order Traversal	
98		Recursive Creation of Binary Tree	
99		Program to find Node Count in the Binary Tree	
100		Program to find count of nodes having 1 child, 2 children and leaf nodes	
101		Program to Find the height of the Binary Tree	
102	Lab 26	write a program or function to find the sum all nodes in a given binary tree.	
103		Program to Find if the given Binary Tree is complete	
104		Program to find if the given Binary Tree is strictly	
105		Program for Level Order Traversal	
106	Lab 27	Write a program to create a copy of the given Binary Tree	
107		Program to build the Expression Tree from the given Infix expression	
108		write a program to check if the given tree is BST or not.	
109		write a program to implement Insertion and Search operation in BST (Iterative)	
110	Lab 28	Program to find the diameter of the Binary Tree (distance between the farthest node)	
111		write a program to implement deletion in BST.	
112		Write a Program for BST insertion (using Recursion)	
113		write a program to perform insertion operation for AVL tree.	
114	Lab 29	Program for Heap Sort	
115		Program for Heap Implementation of Priority Queue	
116		Program for BFS on a Graph	



117		Program for DFS on a Graph	
118	Lab 30	Program to find the number of connected components in the undirected Graph	
119		Program for Warshall's Algorithm for APSP	
120		Program For Linked List Implementation of General Sparse Matrix	

#### Experiment 1,2,4

**Object:** Write an ALGORITHM for array insertion, deletion and traversal.

**DOMAIN:** Array

**ALGORITHM** Traverse(A[], N)

**BEGIN:**

    FOR i=1 TO N DO  
        WRITE(A[i])

**END;**

**Time Complexity:** $\Theta(N)$

**Space Complexity:** $\Theta(1)$

**ALGORITHM** Insertion(A[], N, i, key)

**BEGIN:**

    FOR j=N TO i STEP-1 DO  
        A[j+1]=A[j]  
        A[i]=key  
    N=N+1

**END;**

**Time Complexity:** $\Theta(N)$

**Space Complexity:** $\Theta(1)$

**ALGORITHM** Deletion(A[], N, i)

**BEGIN:**

```
    X=A[i]
    FOR j=i+1 TO N DO
        A[j-1]=A[i]
        N=N-1
    RETURN x
```

**END;**

**Time Complexity:** $\Theta(N)$

**Space Complexity:** $\Theta(1)$

## CODE

### 1.Traversal

```
#include<stdio.h>

int main()
{
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024\n");
    int n,a[100];
    printf("enter no. of elements of array ");
    scanf("%d", &n);

    printf("enter elements of array ");
    for(int i=0;i<n;i++)
        scanf("%d", &a[i]);

    printf("traversed array ");
    for(int i=0;i<n;i++)
        printf("%d", a[i]);

    return 0;
}
```

### 2. Insertion

```
#include<stdio.h>

int main()
{
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024\n");
    int m,n,o,a[100];
    printf("enter no. of elements of array ");
    scanf("%d",&n);
```

```

printf("enter elements of array ");
for(int i=0;i<n;i++)
    scanf("%d",&a[i]);

printf("enter no of elements you want to insert ");
scanf("%d",&m);

printf("enter elements you want to insert ");
for(int i=0;i<m;i++)
{
    scanf("%d",&o);
    a[n+i]=o;
}

printf("new array ");
for(int i=0;i<m+n;i++)
    printf("%d",a[i]);
return 0;
}

```

### 3.Deletion

```

#include<stdio.h>

int main()
{
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024\n");
    int n,m,o,b,c=0,a[100];
    printf("enter no. of elements of array ");
    scanf("%d",&n);

    printf("enter elements of array in sorted form ");

```

```
for(int i=0;i<n;i++)
    scanf("%d",&a[i]);

printf("enter no of elements you want to delete ");
scanf("%d",&m);
```

```
printf("enter element you want to delete ");
for(int i=0;i<m;i++)
{
    scanf("%d",&o);
    for(int k=0;k<n;k++)
    {
        if(a[k]==o)
        {
            c+=1;
            for(int p=k;p<n;p++)
            {
                a[p]=a[p+1];
            }
            break;
        }
    }
}
```

```
printf("new array ");
for(int i=0;i<n-c;i++)
    printf("%d",a[i]);
```

```
return 0;

}
```

## Output

1.

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
enter no. of elements of array 7
enter elements of array 1 2 3 4 5 6 7
traversed array 1 2 3 4 5 6 7
```

2.

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
enter no. of elements of array 5
enter elements of array 1 2 3 4 5
enter no of elements you want to insert 3
enter elements you want to insert 6 7 8
new array 1 2 3 4 5 6 7 8
```

3.

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
enter no. of elements of array 6
enter elements of array in sorted form 1 2 3 4 5 6
enter no of elements you want to delete 2
enter element you want to delete 1 4
new array 2 3 5 6
```

### EXPERIMENT No 3

**Object-**Write an ALGORITHM for insertion in sorted array.

**DOMAIN-**Array

**ALGORITHM Sorted(A[], N, key)**

**BEGIN:**

```
    i=0
    WHILE A[i]<key DO
        i=i+1
    RETURN i
```

**END;**

**Time Complexity:** $\Theta(N)$

**Space Complexity:** $\Theta(1)$

**ALGORITHM: INS\_sorted(A[], N ,i, key)**

**BEGIN:**

```
    FOR j=N-1 TO i STEP-1 DO
        A[j+1]=A[j]
    A[i]=key
    N=N+1
```

**END;**

**Time Complexity:** $\Theta(N)$

**Space Complexity:** $\Theta(1)$

### **CODE**

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

```
int main()
```

```
{
```

```
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024\n");
```

```
    int n,p,m,o,a[100]={0};
```

```
    printf("enter no. of elements of array ");
```

```
    scanf("%d",&n);
```

```
    printf("enter elements of array in sorted form ");
```

```
    for(int i=0;i<n;i++)
```

```

scanf("%d",&a[i]);

printf("enter no of elements you want to insert ");
scanf("%d",&m);

printf("enter elements you want to insert ");
for(int k=0;k<m;k++)
{
scanf("%d",&o);
for(int i=0;i<m+n;i++)
{
if(a[i]<=o && a[i+1]>o || a[i]<=o && a[i+1]==0)
{
for(int x=i+1;x<m+n;x++)
{
p=a[x];
a[x]=o;
o=p;
}}}}
printf("new array ");
for(int i=0;i<m+n;i++)
printf("%d",a[i]);
return 0;
}

```

**Output**



Name-Anubhav Singhal \ CS-A \ 2100320120024

enter no. of elements of array 6

enter elements of array in sorted form 1 2 4 5 6 7

enter no of elements you want to insert 4

enter elements you want to insert 3 8 8 9

new array 1 2 3 4 5 6 7 8 8 9

## EXPERIMENT No 5

**Object-**Write an ALGORITHM for missing number in an array.

**DOMAIN-**Array

**ALGORITHM** missing(A[], N, key)

**BEGIN:**

    int b[max+1]={0}

    for(int i=0;i<n;i++)

        b[a[i]]+=1

    for(int i=0;i<=max;i++)

        if(b[i]==0)

            Printf i

**END;**

**Time Complexity:** $\Theta(N)$

**Space Complexity:** $\Theta(1)$

### **CODE**

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

```
int main()
```

```
{
```

```
    printf("Name-Anubhav Singhal \n CS-A \n 2100320120024\n");
```

```
    int n,a[30];
```

```
    printf("enter no. of elements of array ");
```

```
    scanf("%d",&n);
```

```
    printf("enter elements of array ");
```

```
    for(int i=0;i<n;i++)
```

```
        scanf("%d",&a[i]);
```

```
    int max;
```

```

    printf("enter maximum element of array from 0 to oo :");
    scanf("%d",max);

    int b[max+1]={0};
    for(int i=0;i<n;i++)
    {
        b[a[i]]+=1;
    }

    printf("Missing elements are");
    for(int i=0;i<=max;i++)
    {
        if(b[i]==0)
            printf("%d",i);
    }
    return 0;
}

```

## Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
enter no. of elements of array 5
enter elements of array 1 2 3 4 5
enter maximum element of array from 0 to oo :9
Missing elements are0 6 7 8 9

```

## Experiment 6

**Object:-**Write an ALGORITHM to Find the number which is not repeated in Array of integers, others are present for two times.

**DOMAIN:-**Array

**ALGORITHM:** Arr\_func(A[], N)

**BEGIN:**

```
    K=0,c,B[20]
    FOR i=0 TO N DO
        c=0
        FOR j=0 TO N DO
            IF A[j]==A[i] THEN
                c=c+1
            IF c==1 THEN
                B[k++]=A[i]
        FOR i=0 TO k DO
            WRITE(B[i])
```

**END;**

**Time Complexity:** $\Theta(N^2)$

**Space Complexity:** $\Theta(1)$

### **CODE**

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

```
int main()
```

```
{
```

```
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024\n");
```

```
    int n,a[30];
```

```
    printf("enter no. of elements of array ");
```

```
    scanf("%d",n);
```

```
    printf("enter elements of array ");
```

```
    for(int i=0;i<n;i++)
```

```
        scanf("%d",a[i]);
```

```
    int max=a[0];
```

```
for(int i=1;i<n;i++)  
{  
    if(max<a[i])  
    {  
        max=a[i];  
    }  
}
```

```
int b[max+1]={0};  
for(int i=0;i<n;i++)  
{  
    b[a[i]]+=1;  
}
```

```
printf("repeated elements are ");  
for (int i = 0; i < max+1; i++)  
{  
    if(b[i]>1)  
    {  
        printf("%d",i);  
    }  
}
```

```
printf("non repeated elements are ");  
for (int i = 0; i < max+1; i++)  
{  
    if(b[i]==1)  
    {  
        printf("%d",i<<" ");  
    }  
}
```

```
    }  
}  
  
return 0;  
}
```

### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
enter no. of elements of array 9  
enter elements of array 1 1 3 5 5 6 7 8 9  
repeated elements are 1 5  
non repeated elements are 3 6 7 8 9
```

## EXPERIMENT No 7

**Object-Write an ALGORITHM for reversal of array.**

**DOMAIN-Array**

**ALGORITHM reverse(a[], N, key)**

**BEGIN:**

    mid=n/2

    for(int i=0;i<mid;i++)

        int extra=a[i];

        a[i]=a[n-1-i];

        a[n-1-i]=extra;

**END;**

**Time Complexity: $\Theta(N)$**

**Space Complexity: $\Theta(1)$**

### **CODE**

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

```
int main()
```

```
{
```

```
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024\n");
```

```
    int n,a[100];
```

```
    printf("enter no. of elements of array ");
```

```
    scanf("%d",n);
```

```
    printf("enter elements of array ");
```

```
    for(int i=0;i<n;i++)
```

```
        scanf("%d",a[i]);
```

```
int mid=n/2;
for(int i=0;i<mid;i++)
{
    int extra=a[i];
    a[i]=a[n-1-i];
    a[n-1-i]=extra;
}

printf("reversed array ");
for(int i=0;i<n;i++)
    printf("%d",a[i]);

return 0;
}
```

### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
enter no. of elements of array 9
enter elements of array 1 2 3 4 5 6 7 8 9
reversed array 9 8 7 6 5 4 3 2 1
```



## Program 8

### Program for merging of 2 sorted arrays

#### Algorithm

1. Create a new array to hold the merged result
2. Initialize two index variables i and j to point to the first element of the two sorted arrays
3. Compare the elements at the two indices i and j, and add the smaller one to the new merged array
4. Increment the index of the array from which the smaller element was taken
5. Repeat steps 3-4 until all elements of one of the two sorted arrays have been merged into the new array
6. Add the remaining elements of the other sorted array to the new merged array
7. Return the merged array

#### CODE

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

```
void mergeArrays(int arr1[], int n1, int arr2[], int n2, int arr3[]) {  
    int i = 0, j = 0, k = 0;
```

```
    while (i < n1 && j < n2) {  
        if (arr1[i] < arr2[j])  
            arr3[k++] = arr1[i++];  
        else  
            arr3[k++] = arr2[j++];  
    }
```

```
    while (i < n1)  
        arr3[k++] = arr1[i++];
```

```
    while (j < n2)  
        arr3[k++] = arr2[j++];  
}
```

```
int main() {  
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
```

```
int arr1[] = {1, 3, 5, 7};
int n1 = sizeof(arr1) / sizeof(arr1[0]);
int arr2[] = {2, 4, 6, 8};
int n2 = sizeof(arr2) / sizeof(arr2[0]);
int arr3[n1 + n2];

mergeArrays(arr1, n1, arr2, n2, arr3);

printf("Merged array is: ");
for (int i = 0; i < n1 + n2; i++)
    printf("%d ", arr3[i]);

return 0;
}
```

**Output**

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Merged array is: 1 2 3 4 5 6 7 8
```

### Experiment 9.

**Object-** Write an ALGORITHM for Matrix Addition.

**DOMAIN-**Array

**ALGORITHM:** Matrixadd(A[[]], B[[]], M,N)

**BEGIN:**C[M][N]

    FOR i=1 TO M DO

        FOR j=1 TO N DO

            C[i][j]=A[i][j]+B[i][j]

    RETURN C

**END;**

**Time Complexity:**  $\Theta(N^2)$

**Space Complexity:** $\Theta(N^2)$

#### **CODE**

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

```
int main()
```

```
{
```

```
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024\n");
```

```
    int a,b,c,d;
```

```
    printf("enter dimensions of first matrix ");
```

```
    scanf("%d %d",&a,&b);
```

```
    printf("enter dimensions of second matrix ");
```

```
    scanf("%d %d",&c,&d);
```

```
    if(a!=c || b!=d)
```

```
    {
```

```
        printf("Matrix can not be added");
```

```
        return 0;
```

```
    }
```

```
    int m1[a][b],m2[c][d],m3[a][b];
```

```
    printf("enter elements of first matrix ");
```

```
    for(int i=0;i<a;i++)
```

```
    {
```

```
        for(int j=0;j<b;j++)
```

```
            scanf("%d ",m1[i][j]);
```

```
    }
```

```
    printf("enter elements of second matrix ");
```

```
    for(int i=0;i<a;i++)
```

```
    {
```

```

        for(int j=0;j<b;j++)
            scanf("%d ",m2[i][j]);
    }
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
            m3[i][j]=m1[i][j]+m2[i][j];
    }
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
        {
            printf("%d",m3[i][j]);
        }
        printf("\n");
    }

return 0;
}

```

#### OUTPUT

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter dimensions of first matrix 3 3
Enter dimensions of second matrix 3 3
Enter elements of first matrix 1 2 3 4 5 6 7 8 9
Enter elements of second matrix 9 8 7 6 5 4 3 2 1
10 10 10
10 10 10
10 10 10

```

**Object- Write an ALGORITHM for Matrix Substraction.**  
**DOMAIN-Array**

**ALGORITHM: Matrixadd(A[[]], B[[]], M,N)**  
**BEGIN:**C[M][N]  
    FOR i=1 TO M DO  
        FOR j=1 TO N DO  
            C[i][j]=A[i][j]-B[i][j]  
    RETURN C  
**END;**

**Time Complexity:  $\Theta(N^2)$**   
**Space Complexity:  $\Theta(N^2)$**

**CODE**

```
#include <stdio.h>
int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024\n");
    int a,b,c,d;

    printf("enter dimensions of first matrix ");
    scanf("%d %d",&a,&b);

    printf("enter dimensions of second matrix ");
    scanf("%d %d",&c,&d);

    if(a!=c || b!=d)
    {
        printf("Matrix can not be substracted");
        return 0;
    }

    int m1[a][b],m2[c][d],m3[a][b];

    printf("enter elements of first matrix ");
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
            scanf("%d ",m1[i][j]);
    }
    printf("enter elements of second matrix ");
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
            scanf("%d ",m2[i][j]);
```

```

    }
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
            m3[i][j]=m1[i][j]-m2[i][j];
    }
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
        {
            printf("%d",m3[i][j]);
        }
        printf("\n");
    }

return 0;
}

```

#### OUTPUT

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter dimensions of first matrix 3 3
Enter dimensions of second matrix 3 3
Enter elements of first matrix 1 2 3 4 5 6 7 8 9
Enter elements of second matrix 9 8 7 6 5 4 3 2 1
Diffrence of matrices is
-8  -6  -4
-2  0   2
4   6   8

```

#### Experiment 11

**Object-**Write an ALGORITHM for matrix Multiplication.

## DOMAIN-Array

**ALGORITHM: Matrixmultiply(A[[]], M,N, B[[]], P,Q)**

**BEGIN:**

```
    C[M][Q]
    IF N!=P THEN
    FOR i=1 TO M DO
    FOR j=1 TO Q DO
        C[i][j]=0
        FOR k=1 TO N DO
            C[i][j]=C[i][j]+A[i][k]*B[k][j]
        RETURN C
```

**END;**

**Time Complexity:  $\Theta(N^3)$**

**Space Complexity:  $\Theta(N^2)$**

## CODE

```
#include <stdio.h>
int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024\n");
    int a,b,c,d;

    printf("enter dimensions of first matrix ");
    scanf("%d %d",&a,&b);

    printf("enter dimensions of second matrix ");
    scanf("%d %d",&c,&d);

    if(a!=c || b!=d)
    {
        printf("Matrix can not be added");
        return 0;
    }

    int m1[a][b],m2[c][d],m3[a][b];

    printf("enter elements of first matrix ");
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
            scanf("%d ",m1[i][j]);
    }
    printf("enter elements of second matrix ");
    for(int i=0;i<a;i++)
    {
```

```

        for(int j=0;j<b;j++)
            scanf("%d ",m2[i][j]);
    }
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<d;j++)
        {
            int sum=0;
            for(int k=0;k<b;k++)
            {
                sum+=m1[i][k]*m2[k][j];
            }
            m3[i][j]=sum;
        }
    }

    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
        {
            printf("%d",m3[i][j]);
        }
        printf("\n");
    }

    return 0;
}

```

## OUTPUT

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter dimensions of first matrix 3 3
Enter dimensions of second matrix 3 3
Enter elements of first matrix 1 2 3 4 5 6 7 8 9
Enter elements of second matrix 9 8 7 6 5 4 3 2 1
30  24  18
84  69  54
138 114  90

```



## Experiment.12

**Object-** Write an ALGORITHM for Matrix Transposition.

**DOMAIN:**Array

**ALGORITHM:** Matrixtranspose(A[[]], M,N)

**BEGIN:**

```
    B[N][M]
    FOR i=1 TO M DO
        FOR j=1 TO N DO
            B[j][i]=A[i][j]
        RETURN B
```

**END;**

**Time Complexity:**  $\Theta(N^2)$

**Space Complexity:**  $\Theta(N^2)$

**CODE**

```
#include <stdio.h>
int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024\n");
    int a,b;

    printf("enter dimensions of matrix ");
    scanf("%d %d",&a,&b);
    if(a!=b)
    {
        printf(" Transpose cannot be found ");
        return 0;
    }

    int m[a][b];
    printf("enter elements of matrix ");
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
            scanf("%d ",m[i][j]);
    }

    for(int i=0;i<a;i++)
    {
        for(int j=i+1;j<b;j++)
        {
            swap( m[i][j],m[j][i]);
        }
    }
}
```

```
for(int i=0;i<a;i++)
{
    for(int j=0;j<b;j++)
    {
        printf("%d",m[i][j]);
    }
    printf("\n");
}

return 0;
}
```

#### OUTPUT

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter dimensions of matrix 3 3
Enter elements of matrix 1 2 3 4 5 6 7 8 9
Transpose of matrix is
1 4 7
2 5 8
3 6 9
```

#### Experiment 13

**Object-**Write an ALGORITHM for matrix determinant.

## DOMAIN-Array

**ALGORITHM: Matrixdet(A[[]], M,N)**

**BEGIN:**

```
    FOR i=1 TO M DO
        FOR j=1 TO i DO
            determinant = a[0][0] * ((a[1][1]*a[2][2]) - (a[2][1]*a[1][2])) - a[0][1] * (a[1][0]
                * a[2][2] - a[2][0] * a[1][2]) + a[0][2] * (a[1][0] * a[2][1] - a[2][0] * a[1][1]);
```

```
    RETURN determinant
```

**END;**

**Time Complexity:  $\Theta(N^2)$**

**Space Complexity:  $\Theta(1)$**

## CODE

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

```
int main()
```

```
{
```

```
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
```

```
    int a[3][3], i, j;
```

```
    long determinant;
```

```
    printf("Enter the 9 elements of matrix: ");
```

```
    for(i = 0 ; i < 3; i++)
```

```
        for(j = 0; j < 3; j++)
```

```
            scanf("%d", &a[i][j]);
```

```
    printf("\nThe matrix is\n");
```

```
    for(i = 0; i < 3; i++){
```

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

```
        for(j = 0; j < 3; j++)
```

```
            printf("%d\t", a[i][j]);
```

```
    }
```

```
    determinant = a[0][0] * ((a[1][1]*a[2][2]) - (a[2][1]*a[1][2])) - a[0][1] * (a[1][0]
        * a[2][2] - a[2][0] * a[1][2]) + a[0][2] * (a[1][0] * a[2][1] - a[2][0] * a[1][1]);
```

```
    printf("\nDeterminant of 3X3 matrix: %ld", determinant);
```

```
    return 0;
```

```
}
```

## OUTPUT

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the 9 elements of matrix: 7 94 4 2 4 5 6 3 4

The matrix is

7      94      4
2      4       5
6      3       4
Determinant of 3X3 matrix: 2003
```

#### Experiment 14

**Object-**Write an ALGORITHM for matrix transposition without using second matrix.  
**DOMAIN-**Array

**ALGORITHM: Matrixtranspose(A[[]], M,N)**

**BEGIN:**

```
    FOR i=1 TO M DO
        FOR j=1 TO i DO
            temp=A[i][j]
            A[i][j]=A[j][i]
            A[j][i]=temp
        RETURN A
```

**END;**

**Time Complexity:  $\Theta(N^2)$**

**Space Complexity:  $\Theta(1)$**

**CODE**

```
#include <stdio.h>
int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024\n");
    int a,b;

    printf("enter dimensions of matrix ");
    scanf("%d %d",&a,&b);
    if(a!=b)
    {
        printf(" Transpose cannot be found ");
        return 0;
    }

    int m[a][b];
    printf("enter elements of matrix ");
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
            scanf("%d ",m[i][j]);
    }

    for(int i=0;i<a;i++)
    {
        for(int j=i+1;j<b;j++)
        {
            swap( m[i][j],m[j][i]);
        }
    }

    for(int i=0;i<a;i++)
```

```

{
    for(int j=0;j<b;j++)
    {
        printf("%d",m[i][j]);
    }
    printf("\n");
}

return 0;
}

```

#### OUTPUT

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter dimensions of matrix 3 3
Enter elements of matrix 1 2 3 4 5 6 7 8 9
Transpose of matrix is
1  4  7
2  5  8
3  6  9

```

### Experiment 15

**Object-** Write an ALGORITHM For Linear Search.

**DOMAIN-**Searching

**ALGORITHM** Linear\_search(A[], N, key)

**BEGIN:**

FOR i=1 TO N DO

```

        IF A[i]==key THEN
            RETURN i
    RETURN -1
END;

```

**Worst Case Time Complexity:  $O(N)$**

**Best Case Time Complexity:  $\Omega(1)$**

**Space Complexity:  $\Theta(1)$**

#### CODE

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

```
int main()
```

```
{
```

```
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024\n");
```

```
    int m,n,o,a[100];
```

```
    printf("enter no. of elements of array ");
```

```
    scanf("%d",&n);
```

```
    printf("enter elements of array ");
```

```
    for(int i=0;i<n;i++)
```

```
        scanf("%d",&a[i]);
```

```
    int e;
```

```
    printf("enter element to be searched ");
```

```
    scanf("%d",&e);
```

```
    printf("element is located at ");
```

```
    for(int i=0;i<n;i++)
```

```
        if(a[i]==e)
```

```
        {
```

```
            cout<<i<<" ";
```

```
        }
```

```
return 0;  
}
```

### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
enter no. of elements of array 8  
enter elements of array 1 2 3 4 5 6 7 8  
Enter element to be searched 6  
Element is located at 5
```



## Experiment.16

**Object-Write an ALGORITHM for Binary Search.**

**DOMAIN-Searching**

**ALGORITHM Binary\_search(A[], N, key)**

**BEGIN:**

HIGH=N-1

LOW=0

WHILE LOW<=HIGH DO

MID=(LOW+HIGH)/2

IF A[MID]==key THEN

RETURN MID

ELSE

IF key<A[MID] THEN

HIGH=MID-1

ELSE

LOW=MID+1

RETURN -1

**END;**

**Worst Case Time Complexity:  $O(\log N)$**

**Best Case Time Complexity:  $\Omega(1)$**

**Space Complexity:  $\Theta(1)$**

**CODE**

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

```
int main()
```

```
{
```

```
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024\n");
```

```
    int n,o,a[100];
```

```
    printf("enter no. of elements of array ");
```

```
    scanf("%d",&n);
```

```
    printf("enter elements of array ");
```

```
    for(int i=0;i<n;i++)
```

```
        scanf("%d",&a[i]);
```

```
    printf("enter element to be searched ");
```

```
    scanf("%d",&o);
```

```

printf("element is located at index ");
int s=0,e=n-1,mid=(s+e)/2;
while(mid!=0)
{
    if(o==a[mid])
    {
        cout<<mid;
        break;
    }

    if(a[mid]<o)
    {
        s=mid;
    }

    if(a[mid]>o)
    {
        e=mid;
    }
    mid=(s+o)/2;
}
return 0;
}

```

#### OUTPUT

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
enter no. of elements of array 8
enter elements of array 1 2 3 4 5 6 7 8
Enter element to be searched 4
Element is located at index 3

```

### Experiment.17

**Object-** Write an ALGORITHM for Index Sequential Search.  
**DOMAIN-**Searching

**ALGORITHM:** INDsearch(data[N],KEY,index[M][2])

**BEGIN:**

```
FOR i=0 TO M-1 DO
    IF KEY==index[i][1] THEN
        RETURN index[i][0]
    ELSE
        IF KEY < index[i][1] THEN
            high=index[i][0]-1
            Low =index[i-1][0]+1
            BREAK
        FOR i=low TO high DO
            IF KEY ==data[i] THEN
                RETURN i
        RETURN -1
```

**END;**

**Worst Case Time Complexity:**  $O(N/K+K)$

**Best Case Time Complexity:**  $\Omega(1)$

**Space Complexity:**  $\Theta(1)$

**CODE**

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

```
int sequentialSearch(int k, int a[], int n)
```

```
{
    int i = 0;
    while (i < n)
    {
        if (k == a[i])
        {
            break;
        }
        else
```

```

    {
        i++;
    }
}

if (i < n)
{
    return i;
}
else
{
    return -1;
}
}

int main()
{
    printf("Name-Anubhav Singhal \u CS-A \u 2100320120024\n");
    int n,o,a[100];
    printf("enter no. of elements of array ");
    scanf("%d",&n);

    printf("enter elements of array ");
    for(int i=0;i<n;i++)
        scanf("%d",&a[i]);

    printf("enter element to be searched ");
    scanf("%d",&o);

    int index = sequentialSearch(o, a, n);
    if (index > 0)

```

```
{  
    printf("Element is located at index %d ", index);  
}  
else  
{  
    printf("\nValue not found in the array\n");  
}  
return 0;  
}
```

## OUTPUT

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
enter no. of elements of array 8  
enter elements of array 1 2 3 4 5 6 7 8  
Enter element to be searched 4  
Element is located at index 3
```

### Experiment 18

**Object: Program for Hash Table Implementation for Basic Hash Function (Without collisions)**

**DOMAIN-String**

ALGORITHM DivisionHash(Key, TS)

BEGIN:

    NearestPrime(TS)

$h = \text{Key} \% \text{TS}$

    RETURN h

END;

**Time Complexity:  $\theta(1)$**

**Space Complexity:  $\theta(1)$**

ALGORITHM MidsquareHash(Key, TS)

BEGIN:

$L = \text{LengthOfKey}(\text{Key})$

$n = \text{key} * \text{key}$

$x = \text{Ceil}((2 * L - \text{TS}) / 2)$

$n = n / 10^x$

$h = n \% 10^L$

    RETURN h

END;

**Time Complexity:  $\theta(1)$**

**Space Complexity:  $\theta(1)$**

ALGORITHM FoldingHash(Key, TS)

BEGIN:

$L = \text{LengthOfKey}(\text{Key})$

$n = \text{key}$

    Sum=0

    WHILE  $n \geq 0$  DO

$r = n \% \text{TS}$

        Sum=Sum+r

$n = n / \text{TS}$

$h = \text{Sum} \% \text{TS}$

    RETURN h

END;

**Time Complexity:  $\theta(1)$**

**Space Complexity:  $\theta(1)$**

CODE

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

```

#define MAX_SIZE 10
typedef struct Node {
    char *key;
    int value;
    struct Node *next;
} Node;
Node *table[MAX_SIZE];
int hash(char *key) {
    int sum = 0;
    for (int i = 0; i < strlen(key); i++) {
        sum += key[i];
    }
    return sum % MAX_SIZE;
}
void insert(char *key, int value) {
    int index = hash(key);
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->key = key;
    newNode->value = value;
    newNode->next = NULL;
    if (table[index] == NULL) {
        table[index] = newNode;
    } else {
        printf("Collision occurred at index %d\n", index);
    }
}
int search(char *key) {
    int index = hash(key);
    Node *node = table[index];
    while (node != NULL) {
        if (strcmp(node->key, key) == 0) {
            return node->value;
        }
        node = node->next;
    }
    return -1;
}
int main() {
    printf("Name-Anubhav\\CS-A\\2100320120024\n");
    insert("apple", 3);
    insert("banana", 4);
    insert("orange", 5);
    int result = search("banana");
    if (result == -1) {
        printf("Key not found\n");
    }
}

```

```
} else {  
    printf("Value for key 'banana' is %d\n", result);  
}  
return 0;  
}
```

Output

```
Name-Anybhav\CS-A\2100320120024  
Value for key 'banana' is 4
```

#### Experiment 19

**Object:** Program for Hash Table Implementation for Collision Resolution using Linear Probing

**ALGORITHM** LinearProbing (T[ ], N, Key,h)

**BEGIN:**

    i=h



```

        DO
            IF T[i]==key THEN
                RETURN i
            ELSE
                IF T[i]==Blank THEN
                    RETURN -1
                ELSE
                    i=(i+1)%N
            WHILE(1)
END;

```

**Time Complexity:  $\theta(N)$**

**Space Complexity:  $\theta(1)$**

### **CODE**

```

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

```

```

#define MAX_SIZE 10

```

```

typedef struct Node {
    char *key;
    int value;
} Node;

```

```

Node *table[MAX_SIZE];
int hash(char *key) {
    int sum = 0;
    for (int i = 0; i < strlen(key); i++) {
        sum += key[i];
    }
    return sum % MAX_SIZE;
}

```

```

void insert(char *key, int value) {
    int index = hash(key);
    int i = 0;
    while (table[(index + i) % MAX_SIZE] != NULL) {
        if (strcmp(table[(index + i) % MAX_SIZE]->key, key) == 0) {
            table[(index + i) % MAX_SIZE]->value = value;
            return;
        }
        i++;
    }
}

```

```

Node *newNode = (Node *)malloc(sizeof(Node));
newNode->key = key;
newNode->value = value;
table[(index + i) % MAX_SIZE] = newNode;
}

int search(char *key) {
    int index = hash(key);
    int i = 0;
    while (table[(index + i) % MAX_SIZE] != NULL) {
        if (strcmp(table[(index + i) % MAX_SIZE]->key, key) == 0) {
            return table[(index + i) % MAX_SIZE]->value;
        }
        i++;
    }
    return -1;
}

int main() {
    printf("Name-Anubhav\\CS-A\\2100320120024\\n");
    insert("apple", 3);
    insert("banana", 4);
    insert("orange", 5);
    // Insert a new key-value pair with a collision
    insert("lemon", 6);
    int result = search("banana");
    if (result == -1) {
        printf("Key not found\\n");
    } else {
        printf("Value for key 'banana' is %d\\n", result);
    }
    result = search("lemon");
    if (result == -1) {
        printf("Key not found\\n");
    } else {
        printf("Value for key 'lemon' is %d\\n", result);
    }
    return 0;
}

```

### Output

```

Name-Anubhav\\CS-A\\2100320120024\\n
Value for key 'banana' is 4
Value for key 'lemon' is 6

```

### Experiment 20

Program for Hash Table Implementation for Collision Resolution using Quadratic Probing

**ALGORITHM QuadraticProbing** (T[ ], N, Key,h)

**BEGIN:**

    i=0

    x=h

    DO

        IF T[x]==key THEN

            RETURN x

    ELSE

        IF T[x]==Blank THEN

            RETURN -1

        ELSE

$x=(h+a*i+bi^2)\%N$

```

                                i=i+1
        WHILE(1)
END;
```

Program for Hash Table Implementation for Collision Resolution using Double Hashing/Rehashing

**Time Complexity:  $\theta(N)$**

**Space Complexity:  $\theta(1)$**

### CODE

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_SIZE 10
typedef struct Node {
    char *key;
    int value;
} Node;
Node *table[MAX_SIZE];
int hash(char *key) {
    int sum = 0;
    for (int i = 0; i < strlen(key); i++) {
        sum += key[i];
    }
    return sum % MAX_SIZE;
}

void insert(char *key, int value) {
    int index = hash(key);
    int i = 0;
    while (table[(index + i * i) % MAX_SIZE] != NULL) {
        if (strcmp(table[(index + i * i) % MAX_SIZE]->key, key) == 0) {
            table[(index + i * i) % MAX_SIZE]->value = value;
            return;
        }
        i++;
    }
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->key = key;
    newNode->value = value;
    table[(index + i * i) % MAX_SIZE] = newNode;
}

int search(char *key) {
    int index = hash(key);
    int i = 0;
```

```

while (table[(index + i * i) % MAX_SIZE] != NULL) {
    if (strcmp(table[(index + i * i) % MAX_SIZE]->key, key) == 0) {
        return table[(index + i * i) % MAX_SIZE]->value;
    }
    i++;
}
return -1;
}

int main() {
    printf("Name-Anubhav\\CS-A\\2100320120024\\n");
    insert("apple", 3);
    insert("banana", 4);
    insert("orange", 5);
    // Insert a new key-value pair with a collision
    insert("lemon", 6);
    int result = search("banana");
    if (result == -1) {
        printf("Key not found\\n");
    } else {
        printf("Value for key 'banana' is %d\\n", result);
    }
    result = search("lemon");
    if (result == -1) {
        printf("Key not found\\n");
    } else {
        printf("Value for key 'lemon' is %d\\n", result);
    }
    return 0;
}

```

### Output

```

Name-Anubhav\\CS-A\\2100320120024
Value for key 'banana' is 4
Value for key 'lemon' is 6

```

### Experiment 21

**ALGORITHM DoubleHashingProbe (T[ ], N, Key,h,h')**

**BEGIN:**

    i=1

    x=h

    DO

        IF T[x]==key THEN

            RETURN x

        ELSE

            IF T[x]==Blank THEN

                RETURN -1

            ELSE

                x=(h+i\*h')%N

                i=i+1

    WHILE(1)

**END;**

**Time Complexity:  $\theta(N)$**

**Space Complexity: $\theta(1)$**

### **CODE**

#include <stdio.h>

#include <stdlib.h>

```

#include <string.h>
#define MAX_SIZE 10
typedef struct Node {
    char *key;
    int value;
} Node;
Node *table[MAX_SIZE];
int hash(char *key) {
    int sum = 0;
    for (int i = 0; i < strlen(key); i++) {
        sum += key[i];
    }
    return sum % MAX_SIZE;
}
int hash2(char *key) {
    int sum = 0;
    for (int i = 0; i < strlen(key); i++) {
        sum += key[i];
    }
    return (7 - (sum % 7));
}
void insert(char *key, int value) {
    int index = hash(key);
    int i = 0;
    int step = hash2(key);
    while (table[index] != NULL) {
        if (strcmp(table[index]->key, key) == 0) {
            table[index]->value = value;
            return;
        }
        i++;
        index = (index + step * i) % MAX_SIZE;
    }
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->key = key;
    newNode->value = value;
    table[index] = newNode;
}
int search(char *key) {
    int index = hash(key);
    int i = 0;
    int step = hash2(key);
    while (table[index] != NULL) {
        if (strcmp(table[index]->key, key) == 0) {
            return table[index]->value;
        }
    }
}

```

```

    }
    i++;
    index = (index + step * i) % MAX_SIZE;
}
return -1;
}

int main() {
    printf("Name-Anubhav\\CS-A\\2100320120024\\n");
    insert("apple", 3);
    insert("banana", 4);
    insert("orange", 5);
    // Insert a new key-value pair with a collision
    insert("lemon", 6);
    int result = search("banana");
    if (result == -1) {
        printf("Key not found\\n");
    } else {
        printf("Value for key 'banana' is %d\\n", result);
    }
    result = search("lemon");
    if (result == -1) {
        printf("Key not found\\n");
    } else {
        printf("Value for key 'lemon' is %d\\n", result);
    }
    return 0;
}

```

## Output

```

Name-Anubhav\\CS-A\\2100320120024
Value for key 'banana' is 4
Value for key 'lemon' is 6

```



### **Experiment.22**

**Q. Write an ALGORITHM to calculate factorial of a number using recursion.**

**ALGORITHM FACTORIAL(a)**

**BEGIN :**

IF a==0

    RETURN(1)

ELSE

    IF(a>0)

        RETURN(a\*FACTORIAL(a-1))

**END;**

**Time Complexity:  $\Theta(n)$**

**Space Complexity:  $\Theta(n)$**

**CODE**

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

```
int fact(int a)
```

```
{
```

```
    int b=1;
```

```
    if(a==0)
```

```
    {
```

```
        return 1;
```

```
    }
```

```
    else if(a==1)
```

```
    {
```

```
        return 1;
```

```
    }
```

```
    else
```

```
    {
```

```
        int b=a*fact(a-1);
```

```
        return b;
```

```
    }
```

```

}
int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    int n;
    printf(" Enter number ");
    scanf("%d",&n);
    printf("%d",&fact(n));
    return 0;
}

```

#### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter number 5
120

```

#### Experiment.23

**Q. Write an ALGORITHM to calculate power of a given number using recursion.**

ALGORITHM POWER(a,b)

BEGIN:

    IF b==0 THEN

        RETURN 1

    ELSE

        RETURN a\*POWER(a,b-1)

END;

**Time Complexity:  $O(b)$**

**Space Complexity:  $\Theta(b)$**

#### CODE

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

```
int power(int a,int p)
```

```

{
    if(p==0)
    {
        return 1;
    }
    else
    {
        int b=a*power(a,p-1);
        return b;
    }
}

```

```
int main()
```

```

{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    int n,p;
    printf(" Enter number ");

```

```

    scanf("%d",&n);
    printf(" Enter power ");
    scanf("%d",&p);
    cout<<power(n,p);
    return 0;
}

```

#### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter number 4
Enter power 4
256

```

#### Experiment.24

**Q. Write an ALGORITHM to Calculate n terms of Fibonacci series using recursion.**

ALGORITHM Fibo(a)

BEGIN:

```

    IF a==1 THEN
        RETURN 0
    ELSE
        IF a==2 THEN
            RETURN 1
        ELSE
            RETURN Fibo(a-1)+Fibo(a-2)

```

END;

**Time Complexity:  $\Theta(2^n)$**

**Space Complexity:  $\Theta(N)$**

**CODE**

```

#include <stdio.h>
int fib(int a)
{
    if(a==1)
    {
        return 0;
    }
    else if(a==2)
    {
        return 1;
    }
    return fib(a-1)+fib(a-2);
}
int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    int n;

```

```

printf(" Enter number ");
scanf("%d",&n);
printf("%d",fib(n));
return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter number 5
3

```

### Experiment.25

**Q. Program for finding GCD of two numbers using Recursion.**

**ALGORITHM gcd(int a, int b)**

**BEGIN**

```

if(a==0 || b==0)
    Print can not be found

```

```

if(a==b)
    return a

```

```

else if(a>b)
    return gcd(a-b,b);

```

```

else if(b>a)
    return gcd(a,b-a);

```

**END;**

**Time Complexity:  $\Theta(n)$**

**Space Complexity:  $\Theta(n)$**

### CODE

```

#include<stdio.h>
int gcd(int a,int b)
{
    if(a==0 || b==0)
    {
        exit(0);
    }

    if(a==b)
    {
        return a;
    }
}

```

```

else if(a>b)
{
    return gcd(a-b,b);
}

else if(b>a)
{
    return gcd(a,b-a);
}
}

int main()
{
    int a,b;
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    printf("Enter                the                number:                ");
    scanf("%d %d", &a,&b);
    printf("%d",gcd(a,b));
    return 0;
}

```

#### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter 2 numbers 3 9
3

```

### Experiment.26

**Q. Program for finding GCD of two numbers using Recursion.**

**ALGORITHM gcd(int a, int b)**

**BEGIN**

if(a==0 || b==0)

Print can not be found

if(a==b)

return a

else if(a>b)

return gcd(a-b,b);

else if(b>a)

return gcd(a,b-a);

**END;**

**Time Complexity:  $\Theta(n)$**

**Space Complexity:  $\Theta(n)$**

**CODE**

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

```
int bs(int b[],int e,int m,int n)
```

```
{
```

```
int mid=(m+n)/2;
```

```
if(b[mid]>e)
```

```
{
```

```
bs(b,e,0,mid-1);
```

```
}
```

```
if(b[mid]<e)
```

```
{
```

```
bs(b,e,mid+1,n);
```

```

    }

    return mid;
}

int main()
{
    cout<<"Name-Anubhav Singhal \ CS-A \ 2100320120024"<<endl;
    int n,a[100];
    printf("Enter          the          total          number          of          elements          ");
    scanf("%d", &n);

    printf("Enter the elements of list");
    for(int i=0;i<n;i++)
        scanf("%d", &a[i]);

    int o;
    cout<<"Enter element to be searched ";
    printf("%d",o);

    printf(" Element is located at index ");
    bs(a,o,0,n);
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024

Enter the total number of elements 8

Enter the elements of list :1 2 3 4 5 6 7 8

Enter element to be searched : 3

Number present at 3

```

### Experiment.27

**Q. Write an ALGORITHM for tower of Hanoi for n disk.**

```
ALGORITHM TOH(N,S,M,D)
BEGIN:
IF N==1 THEN
    Transfer disk from S to D
ELSE
    TOH(N-1,S,M,D)
    Transfer Disk From S to D
    TOH(N-1M,S,D)
End;
```

**Time Complexity:  $\Theta(2^n)$**

**Space Complexity:  $\Theta(n)$**

#### CODE

```
#include<stdio.h>
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
{
    if (n == 1)
    {
        printf("\n Move disk 1 from rod %c to rod %c", from_rod, to_rod);
        return;
    }
    towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
    printf("\n Move disk %d from rod %c to rod %c", n, from_rod, to_rod);
    towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
}
int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    int n = 3;
    towerOfHanoi(n, 'A', 'C', 'B');
    return 0;
}
```

#### Output



```
Name-Anubhav Singhal \ CS-A \ 2100320120024
```

```
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
```

### Experiment.28

**Q. Write an ALGORITHM to calculate reverse of a number using recursion.**

ALGORITHM REV (a,len)

BEGIN:

    IF len ==1

        RETURN a

    ELSE

        RETURN((a%10)\*pow(10,len-1))+REV(a/10,len-1)

END;

**Time Complexity:  $\Theta(\log n)$**

**Space Complexity:  $\Theta(\log n)$**

### CODE

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

```
int sum=0,rem;
```

```
int reverse_function(int num)
```

```
{
```

```
    if(num)
```

```
    {
```

```
        rem=num%10;
```

```
        sum=sum*10+rem;
```

```
        reverse_function(num/10);
```

```
    }
```

```
    else
```

```
        return sum;
```

```
    return sum;
```

```
}
```

```
int main()
```

```
{
```

```
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
```

```
    int
```

```
        num,reverse_number;
```

```
    printf("Enter any number:");
```

```
    scanf("%d",&num);
```

```
reverse_number=reverse_function(num);  
printf("The reverse of entered number is :%d",reverse_number);  
return 0;  
}
```

#### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
Enter any number:123456789  
The reverse of entered number is :987654321
```

### Experiment.29

Q. Finding sum of the digits of the number.

ALGORITHM sum (int num)

```
BEGIN
    if (num != 0)
        return (num % 10 + sum (num / 10));
    else
        return 0;
END;
```

Time Complexity:  $\Theta(n)$

Space Complexity:  $\Theta(n)$

#### CODE

```
#include <stdio.h>
int sum (int num)
{
    if (num != 0)
    {
        return (num % 10 + sum (num / 10));
    }
    else
    {
        return 0;
    }
}

int main()
{
    int num, result;
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024 \n");
    printf("Enter the number: ");
    scanf("%d", &num);
    result = sum(num);
    printf("Sum of digits in %d is %d\n", num, result);
    return 0;
}
```

#### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the number: 5573
Sum of digits in 5573 is 20
```

### Experiment 30

**Object-**Write an **ALGORITHM** to find IF the given string is palindrome or not.  
**DOMAIN-**String

**ALGORITHM** StringPalindrome( String str[ ] )

**BEGIN:**

```
L = Length Of String( str[ ] )
i = 0
WHILE i != L/2
    IF str[ i ] = str[ L - i ]
        c = c + 1
    IF c == L/2
        RETURN TRUE
    ELSE
        RETURN FALSE
```

**END;**

**Time Complexity:**  $\theta(L)$

**Space Complexity:**  $\theta(1)$

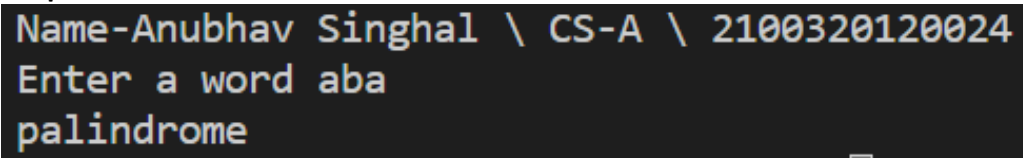
**CODE**

```
#include <stdio.h>
#include <string.h>
void check(char word[], int index)
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    int len = strlen(word) - (index + 1);
    if (word[index] == word[len])
    {
        if (index + 1 == len || index == len)
        {
            printf("palindrome\n");
            return;
        }
        check(word, index + 1);
    }
    else
    {
        printf(" not a palindrome\n");
    }
}

int main()
{
    char word[15];
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    printf("Enter a word ");
    scanf("%s", word);
```

```
    check(word, 0);  
    return 0;  
}
```

#### Output

A screenshot of a terminal window with a black background and white text. The first line shows the user's name and ID: "Name-Anubhav Singhal \ CS-A \ 2100320120024". The second line shows the prompt "Enter a word aba". The third line shows the output "palindrome".

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
Enter a word aba  
palindrome
```

**Object- Write an ALGORITHM for Bubble, Selection and Insertion Sort.**  
**DOMAIN:Sorting**

**ALGORITHM: BubbleSort(A[], N)**

**BEGIN:**

```
    FOR i=1 TO N-1 DO
        FOR j=1 TO N-i DO
            IF A[j]>A[j+1]
                k=A[j]
                A[j]=A[j+1]
                A[j+1]=k
```

**END;**

**Worst Case Time Complexity:** $O(N^2)$

**Best Case Time Complexity:**  $\Omega(N)$

**Space Complexity:** $\Theta(1)$

**CODE**

```
#include<stdio.h>
int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024\n");
    int n,o,a[100];
    printf("enter no. of elements of array ");
    scanf("%d",&n);

    printf("enter elements of array ");
    for(int i=0;i<n;i++)
        scanf("%d",&a[i]);

    for(int i=0;i<n-1;i++)
    {
        for(int j=0;j<n-i;j++)
        {
            if(a[j]>a[j+1])
            {
                swap(a[j+1],a[j]);
            }
        }
    }
}
for(int i=0;i<n;i++)
    printf("%d",a[i]);
return 0;
}
```

**OUTPUT**

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
enter no. of elements of array 5
enter elements of array 3 1 5 4 2
1 2 3 4 5
```

**ALGORITHM: InsertionSort(A[], N)**

**BEGIN:**

```
    FOR i=2 TO N DO
        key=A[i]
        j=i-1
        WHILE j>=1 AND A[j]>key DO
            A[j+1]=A[j]
            j=j-1
        A[j+1]=key
```

**END;**

**Worst Case Time Complexity:** $O(N^2)$

**Best Case Time Complexity:**  $\Omega(N)$

**Space Complexity:** $\Theta(1)$

**CODE**

```
#include<stdio.h>
int main()
{
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024\n");
    int n,a[100];
    printf("enter no. of elements of array ");
    scanf("%d",&n);

    printf("enter elements of array ");
    for(int i=0;i<n;i++)
        scanf("%d",&a[i]);

    int k;
    for(int i=1;i<n;i++)
    {
        k=a[i];
        int j=i-1;
        while(j>=0 && a[j]>k)
        {
            swap(a[j+1],a[j]);
            j--;
        }
    }
```

```
a[j+1]=k;
```

```
for(int i=0;i<n;i++)  
    printf("%d",a[i]);  
return 0;  
}
```

#### OUTPUT

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
enter no. of elements of array 5  
enter elements of array 3 1 5 4 2  
1 2 3 4 5
```

#### ALGORITHM: SelectionSort(A[], N)

##### BEGIN:

```
    FOR i=1 TO N-1 DO  
        min=i  
        FOR j=i+1 TO N DO  
            IF A[j]<A[min] THEN  
                min=j  
        Exchange(A[min], A[i])
```

##### END;

**Time Complexity:  $\Theta(N^2)$**

**Space Complexity:  $\Theta(1)$**

#### CODE

```
#include<stdio.h>  
int main()  
{  
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024\n");  
    int n,a[100];  
    printf("enter no. of elements of array ");  
    scanf("%d",&n);  
  
    printf("enter elements of array ");  
    for(int i=0;i<n;i++)  
        scanf("%d",&a[i]);  
  
    for(int i=0;i<n;i++)  
    {  
        for(int j=i+1;j<n;j++)
```



```

    {
        if(a[i]>a[j])
        {
            swap(a[i],a[j]);
        }
    }
}

```

```

for(int i=0;i<n;i++)
    printf("%d",a[i]);
return 0;
}

```

#### OUTPUT

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
enter no. of elements of array 5
enter elements of array 3 1 5 4 2
1 2 3 4 5

```

#### Experiment No 34

**Object-** Write an ALGORITHM for Shell Sort.

**DOMAIN:**Sorting

**ALGORITHM** shellSort(array, n):

```
gap = n // 2
WHILE gap > 0 DO
    FOR i=gap to n DO
        temp = array[i]
        j = i
        WHILE j >= gap AND array[j - gap] > temp DO
            array[j] = array[j - gap]
            j -= gap
```

```
        array[j] = temp
    gap //= 2
```

**Time Complexity:  $\Theta(N^2)$**

**Space Complexity:  $\Theta(1)$**

### CODE

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

```
// Shell sort
```

```
void shellSort(int array[], int n) {
    for (int interval = n / 2; interval > 0; interval /= 2) {
        for (int i = interval; i < n; i += 1) {
            int temp = array[i];
            int j;
            for (j = i; j >= interval && array[j - interval] > temp; j -= interval) {
                array[j] = array[j - interval];
            }
            array[j] = temp;
        }
    }
}
```

```
void printArray(int array[], int size) {
    for (int i = 0; i < size; ++i) {
        printf("%d ", array[i]);
    }
    printf("\n");
}
```

```
// Driver code
```

```
int main() {
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024\n");
    int n,a[100];
    printf("enter no. of elements of array ");
    scanf("%d",&n);

    printf("enter elements of array ");
    for(int i=0;i<n;i++)
        scanf("%d",&a[i]);
```

```
shellSort(a,n);  
printArray(data, size);  
}
```

#### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
enter no. of elements of array 5  
enter elements of array 3 1 5 4 2  
1 2 3 4 5
```

#### **Program 35** **Program for Merge Sort** **Algorithm**

1. Define the merge sort function, which takes in an array of integers, the starting index, and the ending index as parameters.
2. Within the merge sort function, check if the starting index is less than the ending index. If it is not, return.

3. Calculate the middle index of the array by taking the average of the starting and ending indices, and rounding down if necessary.
4. Call the merge sort function recursively with the left half of the array, from the starting index to the middle index.
5. Call the merge sort function recursively with the right half of the array, from the middle index + 1 to the ending index.
6. Define a temporary array to hold the merged elements.
7. Merge the two sorted subarrays by iterating through them simultaneously, and comparing the elements to determine the order in which they should be added to the temporary array.
8. Copy the temporary array back to the original array, from the starting index to the ending index.
9. Return from the merge sort function.

### CODE

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

```
void merge(int arr[], int l, int m, int r) {  
    int i, j, k;  
    int n1 = m - l + 1;  
    int n2 = r - m;
```

```
    int L[n1], R[n2];
```

```
    for (i = 0; i < n1; i++) {  
        L[i] = arr[l + i];  
    }  
    for (j = 0; j < n2; j++) {  
        R[j] = arr[m + 1 + j];  
    }
```

```
    i = 0;  
    j = 0;  
    k = l;
```

```
    while (i < n1 && j < n2) {  
        if (L[i] <= R[j]) {  
            arr[k] = L[i];  
            i++;  
        } else {  
            arr[k] = R[j];  
            j++;  
        }  
        k++;  
    }
```

```
}
```

```
while (i < n1) {  
    arr[k] = L[i];  
    i++;  
    k++;  
}
```

```
while (j < n2) {  
    arr[k] = R[j];  
    j++;  
    k++;  
}  
}
```

```
void mergeSort(int arr[], int l, int r) {  
    if (l < r) {  
        int m = l + (r - l) / 2;  
  
        mergeSort(arr, l, m);  
        mergeSort(arr, m + 1, r);  
  
        merge(arr, l, m, r);  
    }  
}
```

```
void printArray(int arr[], int size) {  
    int i;  
    for (i = 0; i < size; i++) {  
        printf("%d ", arr[i]);  
    }  
    printf("\n");  
}
```

```
int main() {  
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024 \n");  
    int arr[] = { 12, 11, 13, 5, 6, 7 };  
    int arr_size = sizeof(arr) / sizeof(arr[0]);  
  
    printf("Given array is \n");  
    printArray(arr, arr_size);  
  
    mergeSort(arr, 0, arr_size - 1);  
}
```

```
printf("\nSorted array is \n");  
printArray(arr, arr_size);  
return 0;  
}
```

### Output

Name-Anubhav Singhal \ CS-A \ 2100320120024

Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13

### Experiment 36

**Object-**Write an ALGORITHM for Quick Sort.

**DOMAIN-**Sorting

**ALGORITHM:** QuickSort(A[],low,high)

**BEGIN:**

```
    IF low<high THEN  
        j=Partition(A[],low,high)  
        QuickSort(A[],low,j-1)  
        QuickSort(A[],j+1,high)
```

**END;**

**ALGORITHM: Partition(A[],low,high)****BEGIN:**

```
i=low, j=high+1,pivot=A[low]
DO
    DO
        i=i+1
    WHILE(A[i]<pivot)
    DO
        J=j-1
    WHILE(A[j]>pivot)

    IF i<j THEN
        Exchange(A[i],A[j])
WHILE(i<j)

    Exchange(A[j],A[low])
RETURN j
```

**END;****Worst Case Time Complexity:** $O(N^2)$ **Best Case Time Complexity:**  $\Omega(N\log_2 N)$ **Space Complexity:**  $\Theta(\log_2 N)$ **CODE**

#include&lt;stdio.h&gt;

int partition(int A[],int low,int high)

```
{
    int i=low;
    int j=high+1;
    int pivot=A[low];
    do
    {
        do
        {
            i++;
        } while (A[i]<pivot);
```

```
do
{
    j--;
} while (A[j]>pivot);
```

```
if(i<j)
    swap(A[i],A[j]);
```

```
if(i>j)
```

```

        swap(A[j],A[low]);

    } while(i<j);
    return j;
}

void quick_sort(int A[],int low,int high)
{
    if(low<high)
    {
        int j=partition(A,low,high);
        quick_sort(A,low,j-1);
        quick_sort(A,j+1,high);
    }
}

int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024\n");
    int n,a[100];
    printf("enter no. of elements of array ");
    scanf("%d",&n);

    printf("enter elements of array ");
    for(int i=0;i<n;i++)
        scanf("%d",&a[i]);

    quick_sort(a,0,n-2);

    for(int i=0;i<n;i++)
        printf("%d",a[i]);
    return 0;
}

```

#### OUTPUT

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
enter no. of elements of array 5
enter elements of array 3 1 5 4 2
1 2 3 4 5

```



### Experiment 37

**Object-**Write an ALGORITHM for Randomised Quick Sort.  
**DOMAIN-**Sorting

**ALGORITHM:** QuickSort(A[],low,high)

**BEGIN:**

    IF low<high THEN

        x=rand()%high+low;

        swap(A[low],A[x]);

        j=Partition(A[],low,high)

        QuickSort(A[],low,j-1)

        QuickSort(A[],j+1,high)

**END;**

**ALGORITHM: Partition(A[],low,high)**

**BEGIN:**

```
    i=low, j=high+1,pivot=A[low]
    DO
        DO
            i=i+1
        WHILE(A[i]<pivot)
        DO
            j=j-1
        WHILE(A[j]>pivot)

        IF i<j THEN
            Exchange(A[i],A[j])
    WHILE(i<j)

    Exchange(A[j],A[low])
    RETURN j
```

**END;**

**Worst Case Time Complexity:** $O(N^2)$

**Best Case Time Complexity:**  $\Omega(N\log_2 N)$

**Space Complexity:**  $\Theta(\log_2 N)$

**CODE**

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

```
int partition(int A[],int low,int high)
```

```
{
    int x=rand()%high+low;
    swap(A[low],A[x]);
    int i=low;
    int j=high+1;
    int pivot=A[low];
    do
    {
        do
        {
            i++;
        } while (A[i]<pivot);
```

```
        do
        {
            j--;
        } while (A[j]>pivot);
```

```
    if(i<j)
        swap(A[i],A[j]);
```

```

        if(i>j)
            swap(A[j],A[low]);

    } while(i<j);
    return j;
}

void quick_sort(int A[],int low,int high)
{
    if(low<high)
    {
        int j=partition(A,low,high);
        quick_sort(A,low,j-1);
        quick_sort(A,j+1,high);
    }
}

int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024\n");
    int n,a[100];
    printf("enter no. of elements of array ");
    scanf("%d",&n);

    printf("enter elements of array ");
    for(int i=0;i<n;i++)
        scanf("%d",&a[i]);

    quick_sort(a,0,n-2);

    for(int i=0;i<n;i++)
        printf("%d",a[i]);
    return 0;
}

```

#### OUTPUT

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
enter no. of elements of array 5
enter elements of array 3 1 5 4 2
1 2 3 4 5

```





### Experiment.38

**Object-**Write an ALGORITHM for Counting Sort.

**DOMAIN-**Sorting

**ALGORITHM:** CountingSort(A[],k,n)

**BEGIN:**

```
    FOR i = 0 TO k DO
        c[i] = 0
    FOR j = 0 TO n DO
        c[A[j]] = c[A[j]] + 1
    FOR i = 1 TO k DO
        c[i] = c[i] + c[i-1]
    FOR j = n-1 TO 0 STEP-1 DO
        B[ c[A[j]]-1 ] = A[j]
        c[A[j]] = c[A[j]] - 1
    RETURN B
```

**END;**

**Time Complexity:**  $\Omega(N)$

**Space Complexity:**  $\Theta(N)$

**CODE**

```
#include<stdio.h>
void counting_sort(int A[],int n,int k)
{
    int C[k+1]={0};
    int B[n+1]={0};
    for(int i=0;i<n;i++)
    {
        C[A[i]]+=1;
    }

    for(int i=1;i<=k;i++)
    {
        C[i]+=C[i-1];
    }

    for(int i=n-1;i>=0;i--)
    {
        B[C[A[i]]]=A[i];
        C[A[i]]-=1;
    }

    for(int i=0;i<=n+1;i++)
    {
        A[i-1]=B[i];
    }
}
```

```
}  
}
```

```
int main()  
{  
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024\n");  
    int n,a[100];  
    int highest=a[0];  
  
    printf("enter no. of elements of array ");  
    scanf("%d",&n);  
  
    printf("enter elements of array ");  
    for(int i=0;i<n;i++)  
    {  
        scanf("%d",&a[i]);  
        if(A[i]>highest)  
            highest=a[i];  
    }  
    counting_sort(a,n,highest);  
  
    for(int i=0;i<n;i++)  
        printf("%d",a[i]);  
    return 0;  
}
```

#### OUTPUT

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
enter no. of elements of array 5  
enter elements of array 3 1 5 4 2  
1 2 3 4 5
```

**Object-Write an ALGORITHM for Radix Sort.**

**DOMAIN-Sorting**

**ALGORITHM: RadixSort(A[],N,d)**

**BEGIN:**

    FOR i=1 TO d DO

        Apply counting Sort on A[] at radix i

**END;**

**Time Complexity:  $\Theta(N)$**

**Space Complexity:  $\Theta(N)$**

**CODE**

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

```
int size(int a)
```

```
{
```

```
    int b=0;
```

```
    while(a>0)
```

```
    {
```

```
        b++;
```

```
        a=a/10;
```

```
    }
```

```
    return b;
```

```
}
```

```
void counting_sort(int D[],int A[],int n)
```

```
{
```

```
    int k=9;
```

```
    int C[k+1]={0};
```

```
    int B[n+1]={0};
```

```
    for(int i=0;i<n;i++)
```

```
    {
```

```
        C[A[i]]+=1;
```

```
    }
```

```
    for(int i=1;i<=k;i++)
```

```
    {
```

```
        C[i]+=C[i-1];
```

```
    }
```

```
    for(int i=n-1;i>=0;i--)
```

```
    {
```

```
        B[C[A[i]]]=D[i];
```

```
        C[A[i]]-=1;
```

```
    }
```



```

    for(int i=0;i<=n+1;i++)
    {
        D[i-1]=B[i];
    }
}

```

```

void radix_sort(int A[],int n,int d)
{
    int B[n];
    for(int k=0;k<3;k++)
    {
        for(int j=0;j<n;j++)
        {
            int c=A[j]/pow(10,k);
            B[j]=c%10;
        }
    }
}

```

```

    counting_sort(A,B,n);

```

```

    for(int j=0;j<n;j++)
    {
        cout<<A[j]<<" ";
    }
    cout<<endl;
}
}

```

```

int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024\n");
    int n,a[100];

    printf("enter no. of elements of array ");
    scanf("%d",&n);

    printf("enter elements of array ");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i])
    }
    radix_sort(A,n,0);
}

```

```
for(int i=0;i<n;i++)
    printf("%d",a[i]);
return 0;
}
```

#### OUTPUT

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
enter no. of elements of array 5
enter elements of array 3 1 5 4 2
1 2 3 4 5
```

#### Stack by pointer.h

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

```
# define maxsize 100
```

```
struct stack
{
    int top;
    int data[maxsize];
};
```

```
void initialize(struct stack *s)
{
    s->top=-1;
}
```

```
int emptyness(struct stack *s)
{
    if(s->top== -1)
        return 1;
    return 0;
}
```

```
int fullness(struct stack *s)
{
    if(s->top==maxsize-1)
    {
        return 1;
    }
    return 0;
}
```

```
int push(struct stack *s,int x)
{
    if(fullness(s))
        return -1;
    s->top++;
    s->data[s->top]=x;
    return 1;
}
```

```
int pop(struct stack *s)
{
    int x;
    if(emptyness(s))
    {
        return -1;
    }
}
```

```

    x=s->data[s->top];
    s->top--;
    return x;
}

```

```

int peek(struct stack *s)
{
    if(emptyness(s))
    {
        return -1;
    }
    return s->data[s->top];
}

```

```

void display(struct stack *s)
{
    if(emptyness(s))
    {
        return ;
    }
    printf("%d", s->data[s->top])
    s->top--;
    display(s);
}

```

#### **Experiment 40**

**Write An ALGORITHM for Stack Primitive Operations.**

```

ALGORITHM      Initialize stack(Stack S)
Begin:
    S.TOP=-1
End;

```

**Time Complexity-  $\theta(1)$**

### Space Complexity- $\theta(1)$

```
ALGORITHM      Push(Stack S,key)
Begin:
    IF S.TOP==SIZE-1 THEN
        WRITE("Stack Overflows")
        EXIT (1)
    S.TOP++
    S.ITEM[S.TOP]=key
End;
```

### Time Complexity- $\theta(N)$ , Space Complexity- $\theta(1)$

```
ALGORITHM      Empty (Stack S)
Begin:
    IF S.TOP==-1 THEN
        RETURN TRUE
    ELSE
        RETURN FALSE
End;
```

### Time Complexity- $\theta(1)$ , Space Complexity- $\theta(1)$

```
ALGORITHM      Pop (Stack S,key)
Begin:
    IF EMPTY(S) THEN
        WRITE ("Stack underflows")
        EXIT(1)
    X=S.ITEM[S.TOP]
    S.TOP--
    RETURN X
End;
```

### Time Complexity- $\theta(1)$ , Space Complexity- $\theta(1)$

```
ALGORITHM      STACKTOP (Stack S)
Begin:
    RETURN (S.ITEM[S.TOP])
End;
```

### Time Complexity- $\theta(1)$ , Space Complexity- $\theta(1)$

#### CODE

```
#include<stdio.h>
#include"stack by pointer.h"
#include<string>
int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    struct stack a;
    printf(" enter choice \n 1.initialize \n 2.push \n 3.pop \n 4.peek \n 5.display \n 6.exit ");
```

```

while(1)
{
    printf("enter choice ");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1:
        {
            initialize(&a);
            break;
        }

        case 2:
        {
            printf("enter item");
            cin>>x;
            int i=push(&a,x);
            if(i==-1)
                printf("overflow ");
            else
                printf("item pushed successfully");
            break;
        }

        case 3:
        {
            x=pop(&a);
            if(x==-1)
                printf("overflow ");
            else
                printf("item pushed successfully");
            break;
        }

        case 4:
        {
            int x=peek(&a);
            if(x==-1)
            {
                printf("underflow");
                break;
            }
            printf("topmost element of stack is %d",x);
            break;
        }

        case 5:
        {
            printf("stack is");
            display(&a);
        }
    }
}

```

```
        break;
    }
    case 6:
        exit(0);
    }
}
return 0;
}
```

### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
enter choice
1.initialize
2.push
3.pop
4.peek
5.display
6.exit
enter choice 1

enter choice 2
enter item 3
item pushed successfully

enter choice 2
enter item 4
item pushed successfully

enter choice 3
popped item is4

enter choice 4
topmost element of stack is 3

enter choice 5
stack is :3
```

### Experiment 41

**Write an ALGORITHM for Decimal to any base conversion.**

ALGORITHM    Decimal to AnyBase(n,b)

BEGIN:

```

Stack S
Initialize(s)
WHILE n!=0 DO
    r=n%b
    PUSH(S,r)
    n=n/b
WHILE ! Empty (s) DO
    X=Pop(s)
    Write(x)
END;

```

**Time Complexity-  $\theta(\log N)$**

**Space Complexity-  $\theta(N)$**

#### CODE

```

#include<stdio.h>
#include"stack by pointer.h"
#include<string>

int main()
{
printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    struct stack a;
    struct stack *s=&a;

    int n,b,sum=0;
    printf("Enter the decimal no. ");
    scanf("%d",&n);
    printf(" Enter the base you want to convert into ");
    scanf("%d",&b);
    initialize(s);

    while(n>0)
    {
        int a=n%b;
        push(s,a);
        n=n/b;
    }

    while(!emptytness(s))
    {
        int x=pop(s);
        sum=sum*10+x;
    }
    printf("%d",sum);
}

```



```
    return 0;  
}
```

#### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
Enter the decimal no. 323  
Enter the base you want to convert into 4  
11003
```

#### EXPERIMENT 42

**OBJECT** - Program to check the validity of Parthesized Arithmetic Expression using Stack.

**DOMAIN** - Stack

**ALGORITHM**                      Valid Expression (String Exp[])

**BEGIN:**

    Valid=1

```

Stack S
Initialize S
WHILE Exp[i] != '$' DO
    IF Exp[i] = '(' THEN
        PUSH (S,Exp[i])
    ELSE
        IF Exp[i] = ')' THEN
            IF (Empty (S)) THEN
                Valid=0
                BREAK
            ELSE
                Pop(S)
        i++
        IF valid == 0 THEN
            WRITE("Valid Expression")
        ELSE
            IF Empty(S) THEN
                WRITE("Valid Expression")
            ELSE
                WRITE("Invalid Expression")
END;

```

**TIME COMPLEXITY- $\Theta(n)$**

**SPACE COMPLEXITY- $\Theta(n)$**

**CODE**

```

#include<stdio.h>
#include<math.h>
#include"stack.h"
#include<string>
int main()
{
    printf("Name-Anubhav Singh \ CS-A \ 2100320120024 \n");
    struct stack a;
    string e;
    printf("Enter the expression ");
    scanf("%s",&e);
    e=e+' ';
    initialize(&a);
    int i=0;
    while(e[i]!=' ' && i<=e.length())
    {
        if(e[i]=='(')
        {
            push(&a,e[i]);
            i++;
            while(e[i]!=' ')
            {
                push(e[i]);
            }
        }
    }
}

```

```

        i++;
    }
    if(e[i]=='')
    {
        int a=pop(&a);
        while(a!='(')
        {
            a=pop(&a);
        }
    }
    i++;
    continue;
}
if(e[i]=='')
{
    c
    exit(0);
}
i++;
}
if(emptyness(&a))
{
    printf("correct expression ");
    exit(0);
}
printf("incorrect expression ");
}

```

#### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the expression a+d-(k+d)
Correct expression

```

### EXPERIMENT 43

**OBJECT** - Program to check the validity of Bracked Arithmetic Expression using Stack.

**DOMAIN** - Stack

**ALGORITHM** Valid Expression (String Exp[])

**BEGIN:**

Valid=1

Stack S

```

Initialize S
WHILE Exp[i] != '$' DO
    IF Exp[i] = '[' THEN
        PUSH (S,Exp[i])
    ELSE
        IF Exp[i] = '[' THEN
            IF (Empty (S)) THEN
                Valid=0
                BREAK
            ELSE
                Pop(S)
        i++
        IF valid == 0 THEN
            WRITE("Valid Expression")
        ELSE
            IF Empty(S) THEN
                WRITE("Valid Expression")
            ELSE
                WRITE("Invalid Expression")
END;

```

**TIME COMPLEXITY- $\Theta(n)$**   
**SPACE COMPLEXITY- $\Theta(n)$**

#### CODE

```

#include<stdio.h>
#include<math.h>
#include"stack by pointer.h"
#include<string>

int paranthesis(struct stack *x,string e,int i)
{
    push(x,e[i]);
    i++;
    while(e[i]!='')
    {
        if(e[i]=='{' || e[i]=='}' || e[i]=='[' || e[i]==']')
        {
            printf("incorrect expression ");
            exit(0);
        }
        push(x,e[i]);
        i++;
    }

    if(e[i]=='')
    {

```

```

        int a=pop(x);
        while(a!='(')
        {
            a=pop(x);
        }
        pop(x);
    }
    i++;
    return i;
}

```

```

int curly(struct stack *x,string e,int i)
{
    push(x,e[i]);
    i++;
    while(e[i]!='}')
    {
        if(e[i]=='(')
        {
            i=paranthesis(x,e,i);
        }
        if(e[i]=='[' || e[i]=='{' || e[i]=='')
        {
            printf("incorrect expression ");
            exit(0);
        }
        push(x,e[i]);
        i++;
    }
}

```

```

if(e[i]!='}')
{
    int a=pop(x);
    while(a!='(')
    {
        a=pop(x);
    }
    pop(x);
}
i++;
return i;
}

```

```

int box(struct stack *y,string e,int i)
{
    push(y,e[i]);
    i++;
    while(e[i]!='']')

```

```

{
    if(e[i]=='}' || e[i]==')')
    {
        printf("incorrect expression ");
        exit(0);
    }
    if(e[i]=='(')
    {
        i=paranthesis(y,e,i);
    }
    if(e[i]=='{')
    {
        i=curly(y,e,i);
    }
    push(y,e[i]);
    i++;
}
if(e[i]==']')
{
    int a=pop(y);
    while(a!='(')
    {
        a=pop(y);
    }
    pop(y);
}
i++;
return i;
}

```

```

int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    string e;
    printf("enter the expression ");
    scanf("%s",&e);
    e=e+' ';
    struct stack s;
    struct stack *a=&s;
    initialize(a);
    int i=0;
    while(e[i]!='/' && i<=e.length())
    {
        if(e[i]=='(')
        {
            i=paranthesis(a,e,i);
        }
    }
}

```

```

    if(e[i]=='{')
    {
        i=curly(a,e,i);
    }

    if(e[i]=='[')
    {
        i=box(a,e,i);
    }

    if(e[i]==' ' || e[i]=='}' || e[i]=='}')
    {
        printf("incorrect expression ");
        exit(0);
    }
    i++;
}
if(emptyness(a))
{
    printf("correct expression ");
    exit(0);
}
printf("incorrect expression ");
return 0;
}

```

#### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the expression a+d[
Incorrect expression

```

#### Experiment. 44

**OBJECT-**Write an ALGORITHM to check IF given number is palindrome using stack.

**DOMAIN -**Stack

ALGORITHM palindrome check (str[])

Begin:

i=0

Stack S

```

    Initialize (s)
WHILE str[i] != '\0' DO
    PUSH (s , str[i])
    i++

i=0
WHILE str[i] != '\0' DO
    IF Str[i]==StackTOP(S) THEN
        POP(s)
    ELSE
        Break;
IF Empty (s) THEN
    WRITE ("Palindrome")
ELSE
    WRITE (" Not Palindrome")
End;
TIME COMPLEXITY -  $\Theta(n)$ 
SPACE COMPLEXITY -  $\Theta(n)$ 

```

```

#include<iostream>
#include<math.h>
#include"stack.h"
#include<string>
using namespace std;

```

```

int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    int n1,n2=0,i=0;
    printf("Enter the number ");
    scanf("%d",&n1);
    int a=n1;
    struct stack a;

    initialize(&a);

    while(a>0)
    {
        int b=a%10;
        push(b);
        a=a/10;
    }

    while(!emptiness(&a))
    {

```



```

        int b=pop();
        n2=n2+b*pow(10,i);
        i++;
    }

    if(n1==n2)
    {
        printf("Enter the number ");
    }
    else
    {
        printf("Enter the number ");
    }
    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the number 7869
Not a palindrome no.

```

### Experiment.45

**OBJECT - Write an ALGORITHM to reverse a string using Stack.**

**DOMAIN- Stack**

**ALGORITHM** String Reverse(String str[])

Begin:

```

    i=0
    Stack S
    Initialize (s)
    WHILE str[i] !='\0' DO

```

```

        PUSH (s , str[i])
        i++
        WHILE ! Empty(s) DO
            x=pop(s)
            WRITE(x)
    End;

```

**TIME COMPLEXITY -  $\Theta(1)$**

**SPACE COMPLEXITY -  $\Theta(1)$**

#### **CODE**

```

#include<stdio.h>
#include"stack by pointer.h"
#include<string>

int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    string s;
    printf("Enter the string ");
    scanf("%s",&s);
    int n=s.length();

    struct stack a;
    struct stack *p=&a;
    initialize(p);

    for(int i=0;i<n;i++)
    {
        push(p,s[i]);
    }
    for(int i=0;i<n;i++)
    {
        char c=pop(p);
        printf("%d",c);
    }
    return 0;
}

```

#### **Output**

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the string whfhf
fhfhw

```

## EXPERIMENT 46

**OBJECT:** To write an ALGORITHM for Evaluation postfix expression.

**DOMAIN:** STACK

**ALGORITHM** POSTFIX EVALUATION (Postfix Expression)

BEGIN:

```
    STACK OpndStack
    Initialize (OpndStack)
    WHILE not end of input from postfix expression Do
        Symbol = Next character from postfix expression
        IF Symbol is an operand THEN
            push (OpndStack ,symbol)
        ELSE
            oprnd 2=POP (OpndStack)
            oprnd 1=POP(OpndStack)
            value=Result of applying symbol to oprnd1 and oprnd2
            push(OpndStack,value)
        Result = pop(OpndStack)
    RETURN Result
END;
```

**TIME COMPLEXITY-**  $\Theta(N)$

**SPACE COMPLEXITY-**  $\Theta(N)$

### CODE

```
#include<stdio.h>
#include<math.h>
#include"stack by pointer.h"
#include<math.h>
#include<string>
int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    string e;
    printf("Enter the expression ");
    scanf("%s",&e)
    e=e+'$';
    struct stack s;
    struct stack *a=&s;
    initialize(a);
    int i=0;
    while(e[i]!='$')
    {
        if(e[i]>'0' && e[i]<='9')
```

```

{
    int f=e[i]-'0';
    push(a,f);
    i++;
    continue;
}
if(e[i]=='+')
{
    int p=pop(a);
    int q=pop(a);
    push(a,q+p);
    i++;
    continue;
}
if(e[i]=='-')
{
    int p=pop(a);
    int q=pop(a);
    push(a,q-p);
    i++;
    continue;
}
if(e[i]=='*')
{
    int p=pop(a);
    int q=pop(a);
    push(a,q*p);
    i++;
    continue;
}
if(e[i]=='^')
{
    int p=pop(a);
    int q=pop(a);
    push(a,pow(q,p));
    i++;
    continue;
}
i++;
}
printf("%d",pop(a));
return 0;
}

```

#### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the expression 93+
12

```

## EXPERIMENT 47

**OBJECT:** To write an ALGORITHM for prefix evaluation.

**DOMAIN:** STACK

**ALGORITHM** PREFIX EVALUATION (Prefix Expression)

BEGIN:

Reverse (Prefix Expression)

STACK OpStack

Initialize (OpStack)

WHILE not end of input from prefix expression DO

Symbol = Next character from prefix equation

IF Symbol is an operand THEN

push (OpStack ,symbol)

ELSE

oprnd 1=push(OpStack)

oprnd 2=push(OpStack)

value=Result of applying symbol to oprnd1 and oprnd2

push(OpStack ,value)

Result = pop(OpStack)

RETURN Result

END;

**TIME COMPLEXITY-**  $\Theta(N)$

**SPACE COMPLEXITY-**  $\Theta(N)$

**CODE**

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

```
#include<math.h>
```

```
#include"stack by pointer.h"
```

```
#include<math.h>
```

```
#include<string>
```

```
string reverse(string s)
```

```
{
```

```
    string k="";
```

```
    int n=s.length();
```

```
    struct stack a;
```

```
    struct stack *p=&a;
```

```
    initialize(p);
```

```
    for(int i=0;i<n;i++)
```

```
{
```

```

        push(p,s[i]);
    }

    for(int i=0;i<n;i++)
    {
        char c=pop(p);
        k+=c;
    }
    return k;
}

int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    string e;
    printf("Enter the expression ");
    scanf("%s",&e)
    e=reverse(e);
    e=e+'$';
    struct stack s;
    struct stack *a=&s;
    initialize(a);
    int i=0;
    while(e[i]!='$')
    {
        if(e[i]>'0' && e[i]<='9')
        {
            int f=e[i]-'0';
            push(a,f);
            i++;
            continue;
        }
        if(e[i]=='+')
        {
            int p=pop(a);
            int q=pop(a);
            push(a,q+p);
            i++;
            continue;
        }
        if(e[i]=='-')
        {
            int p=pop(a);
            int q=pop(a);
            push(a,q-p);
            i++;
        }
    }
}

```

```

        continue;
    }
    if(e[i]=='*')
    {
        int p=pop(a);
        int q=pop(a);
        push(a,q*p);
        i++;
        continue;
    }
    if(e[i]=='^')
    {
        int p=pop(a);
        int q=pop(a);
        push(a,pow(q,p));
        i++;
        continue;
    }
    i++;
}
printf("%d",pop(a));
return 0;
}

```

#### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the expression -94
49-
5

```

**OBJECT:** To write an **ALGORITHM** for infix to postfix conversion.

**DOMAIN:** STACK

**ALGORITHM**     Infix to postfix (Infix expression)

BEGIN:

```
    STACK (Opstack)
    Initialize (Opstack)
    WHILE not the end of input from Infix Expression DO
        Symbol = next symbol from Infix Expression
    IF Symbol is an operand THEN
        Add symbol to postfix expression
    ELSE
        WHILE (! Empty (Opstack) && Prcd (Stack Top(Opstack) ,Symbol)
            x=pop (Opstack)
            Add x to postfix Expression

            IF Symbol == ')' THEN
                x = pop(Opstack)
            ELSE
                PUSH( Opstack ,Symbol)
        PUSH( Opstack ,Symbol)

    WHILE ! Empty(Opstack) DO
        x= pop(Opstack)
        Add x to Postfix Expression
    RETURN Postfix Expression
END;
```

**TIME COMPLEXITY-**  $\Theta(N)$

**SPACE COMPLEXITY-**  $\Theta(N)$

**CODE**

```
#include<stdio.h>
#include<stdlib.h>
#include"stack by pointer.h"
#include<string.h>
#define SIZE 100
char stack[SIZE];
int top = -1;

int is_operator(char symbol)
{
    if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-')
    {
        return 1;
    }
}
```



```

    else
    {
        return 0;
    }
}

```

```

int precedence(char symbol)
{
    if(symbol == '^')
    {
        return(3);
    }
    else if(symbol == '*' || symbol == '/')
    {
        return(2);
    }
    else if(symbol == '+' || symbol == '-')
    {
        return(1);
    }
    else
    {
        return(0);
    }
}

```

```

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
    struct stack a;
    int i, j;
    char item;
    char x;

    push(&a, '(');
    strcat(infix_exp, " ");

```

```

    i=0;
    j=0;
    item=infix_exp[i];
    while(item != '\0')
    {
        if(item == '(')
        {
            push(&a, item);
        }
        else if( isdigit(item) || isalpha(item))
        {

```

```

    postfix_exp[j] = item;
    j++;
}
else if(is_operator(item) == 1)
{
    x=pop();
    while(is_operator(x) == 1 && precedence(x)>= precedence(item))
    {
        postfix_exp[j] = x;
        j++;
        x = pop(&a);
    }
    push(&a,x);

    push(&a,item);
}
else if(item == ')')
{
    x = pop(&a);
    while(x != '(')
    {
        postfix_exp[j] = x;
        j++;
        x = pop(&a);
    }
}
else
{
    printf("\nInvalid infix Expression.\n");
    getchar();
    exit(1);
}
i++;

item = infix_exp[i];
}
if(top>0)
{
    printf("\nInvalid infix Expression.\n");
    getchar();
    exit(1);
}
if(top>0)
{
    printf("\nInvalid infix Expression.\n");
    getchar();
    exit(1);
}
}

```

```
    postfix_exp[j] = '\0';  
}  
  
int main()  
{  
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");  
    char infix[SIZE], postfix[SIZE];  
    printf("\nEnter Infix expression : ");  
    gets(infix);  
  
    InfixToPostfix(infix,postfix);  
    printf("Postfix Expression: ");  
    puts(postfix);  
    return 0;  
}
```

#### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
  
Enter Infix expression : a+s  
Postfix Expression: as+
```

## EXPERIMENT 49

**OBJECT:** To write an ALGORITHM for infix to prefix conversion.

**DOMAIN:** STACK

**ALGORITHM**      Infix to postfix (Infix expression)

BEGIN:

```
Reverse (Infix Expression)
STACK (Opstack)
Initialize (Opstack)
WHILE not the end of Symbol from Infix Expression DO
    Symbol = next symbol from Infix Expression
    IF Symbol is an operand THEN
        Add symbol to postfix expression
    ELSE
        WHILE (! Empty (Opstack) && ! Prcd ( Symbol,Stack Top(Opstack) )
            x=pop (Opstack)
            Add x to postfix Expression
        IF Symbol = '=' THEN
            x = pop(Opstack)
        ELSE
            PUSH( Opstack ,Symbol)
        PUSH( Opstack ,Symbol)

    WHILE ! Empty(Opstack) DO
        x= pop(Opstack)
        Add x to Postfix Expression
RETURN Reverse (Prefix Expression)
```

END;

**TIME COMPLEXITY-**  $\Theta(n^2)$

**SPACE COMPLEXITY-**  $\Theta(n)$

**CODE**

```
#include<stdio.h>
#include"stack by pointer.h"
#define max 100
int top=-1, a[max];
```

```
int prcd(char c)
{
    if(c=='')
        return 0;
    else if(c=='+' || c=='-')
        return 1;
    else if(c=='*' || c=='/')
        return 1;
```

```
    return 2;
}
```

```
void infixtoprefix(char infix[max],char prefix[max])
```

```
{
    struct stack s;
    struct stack* a=&s;
    char temp,x;
    int i=0,j=0;
    strrev(infix);
    while(infix[i]!='\0')
    {
        temp=infix[i];
        if(isalnum(temp))
        {
            prefix[j++]=temp;
        }
        else if(temp=='')
            push(a,temp);
        else if(temp=='(')
        {
            while((x=pop(a))!='')
            {
                prefix[j++]=x;
            }
        }
        else
        { while(prcd(a[top])>=prcd(temp))
            {prefix[j++]=pop();}
            push(a,temp);
        }
        i++;
    }
    while(top!= -1)
        prefix[j++]=pop(a);
    prefix[j]='\0';
    strrev(prefix);
}
```

```
main()
```

```
{
    char infix[max],prefix[max];
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    printf("Enter the infix expression\n");
    gets(infix);
    printf("The infix expression is %s\n",infix);
    infixtoprefix(infix, prefix);
}
```

```
printf("The prefix expression is %s\n",prefix);  
return 0;  
}
```

#### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
Enter the infix expression  
a+d+g-r  
The infix expression is a+d+g-r  
The prefix expression is +a+d-gr
```

## EXPERIMENT 50

**OBJECT:** Program for implementation of 2 stacks using a single Array.

**DOMAIN:** STACK

### **ALGORITHM**

BEGIN:

1. Create a array.
2. Divide it into 2 parts.
3. Maxsize of one will be the start of another.
4. Perform all primitive operations by iterative method

END;

**TIME COMPLEXITY-**  $\Theta(1)$

**SPACE COMPLEXITY-**  $\Theta(1)$

### **CODE**

```
#include<stdio.h>
# define maxsize 100
int a[maxsize];
int top1=-1;
int
```

top2=maxsize;

```
int push1(int x)
{
    if(top1==top2-1)
    {
        return -1;
    }
    top1++;
    a[top1]=x;
    return 1;
}
```

```
int push2(int x)
{
    if(top2==top1+1)
    {
        return -1;
    }
    top2--;
    a[top2]=x;
    return 1;
}
```

```
int pop1()
{
    int x;
```

```
    if(top1==-1)
    {
        return -1;
    }
    x=a[top1];
    top1--;
    return x;
}
```

```
int pop2()
{
    int x;
    if(top2==maxsize)
    {
        return -1;
    }
    x=a[top2];
    top2++;
    return x;
}
```

```
int peek1()
{
    if(top1==-1)
    {
        return -1;
    }
    return a[top1];
}
```

```
int peek2()
{
    if(top1==maxsize)
    {
        return -1;
    }
    return a[top2];
}
```

```
void display1()
{
    if(top1==-1)
    {
        return ;
    }
    cout<<a[top1]<<" ";
    top1--;
    display1();
}
```

```
void display2()
{
```



```

    if(top2==maxsize)
    {
        return ;
    }
    cout<<a[top2]<<" ";
    top2++;
    display2();
}

int main()
{
    int choice,x;

    while(1)
    {
        printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
        printf("enter choice \n 1.push1 \n 2.push2 \n 3.pop1 \n 4.pop2 \n 5.peek1 \n 6.peek2 \n
7.display1 \n 8.display2 \n 9.exit");
        printf(" enter choice ");
        scanf("%d",& choice);

        switch(choice)
        {
            case 1:
            {
                printf(" enter item ");
                scanf("%d",&x);
                int i=push1(x);
                if(i==-1)
                    printf(" k overflow");
                else
                    printf(" item pushed successfully ");
                break;
            }

            case 2:
            {
                printf(" enter item ");
                scanf("%d",&x);
                int i=push2(x);
                if(i==-1)
                    printf(" k overflow");
                else
                    printf("item pushed successfully ");
                break;
            }

            case 3:
            {
                x=pop1();
                if(x==-1)

```

```
    printf(" underflow");  
    else  
        printf(" oped item is %d"x);  
    break;  
}
```

```
case 4:  
{  
    x=pop2();  
    if(x==-1)  
        printf(" underflow");  
    else  
        printf(" popped item is %d"x);  
    break;  
}
```

```
case 5:  
{  
    int x=peek1();  
    if(x==-1)  
    {  
        printf(" Stack underflow");  
        break;  
    }  
    printf(" topmost element of stack is %d",x);  
    break;  
}
```

```
case 6:  
{  
    int x=peek2();  
    if(x==-1)  
    {  
        printf(" Stack underflow");  
        break;  
    }  
    printf(" topmost element of stack is %d ",x);  
    break;  
}
```

```
case 7:  
{  
    printf(" stack is :");  
    display1();  
    break;  
}
```

```
case 8:  
{  
    printf(" stack is :");  
    display2();  
    break;  
}
```

```

        case 9:
            exit(0);
            break;
        default:
            printf(" incorrect choice");
            break;
    }
}
return 0;
}

```

## Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
enter choice
1.push1
2.push2
3.pop1
4.pop2
5.peek1
6.peek2
7.display1
8.display2
9.exit
enter choice 1
  enter item 2
item pushed successfully

enter choice 1
  enter item 4
item pushed successfully

enter choice 3
poped item is 4

enter choice 5
topmost element of stack is 2

enter choice 7
stack is :2

enter choice 2
  enter item 4
item pushed successfully

enter choice 4
poped item is 4

enter choice 7
stack is :

```

### Experiment.51

#### Q. Program for Finding Minimum in the Stack.

##### ALGORITHM Minimum()

**BEGIN :**

```
int a=pop()
while(!emptytness())
    int b=pop()
    if(a>b)
        a=b
return a
```

**END;**

**Time Complexity:  $\Theta(n)$**

**Space Complexity:  $\Theta(n)$**

##### CODE

```
#include<stdio.h>
#include"stack.h"
```

```
int minimum()
{
    int a=pop();
    while(!emptytness())
    {
        int b=pop();
        if(a>b)
        {
            a=b;
        }
    }
    return a;
}
```

```
int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    initialize();
    push(3);
    push(4);
    push(1);
    push(6);
    push(7);
    push(2);
    push(3);
    push(8);
    push(9);
```

```
printf("%d",minimum());  
  
return 0;  
}
```

#### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
1
```

## Experiment 52

### ALGORITHM : SORTING STACK

#### BEGIN :

Create a temporary stack that will hold elements in sorted order.

While the original stack is not empty: a. Pop the top element from the original stack and store it in a variable. b. While the temporary stack is not empty and the top element of the temporary stack is greater than the variable, pop the top element from the temporary stack and push it onto the original stack. c. Push the variable onto the temporary stack.

The elements in the temporary stack are now in sorted order, so pop them from the temporary stack and push them onto the original stack.

#### CODE

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

#define MAX_SIZE 100

void sort_stack(int stack[], int size);
void bubble_sort(int arr[], int size);

int main()
{
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    int stack[MAX_SIZE], size, i;

    printf("Enter the size of the stack: ");
    scanf("%d", &size);

    printf("Enter the elements of the stack: ");
    for(i=0; i<size; i++) {
        scanf("%d", &stack[i]);
    }

    sort_stack(stack, size);

    printf("Sorted stack: ");
    for(i=0; i<size; i++) {
        printf("%d ", stack[i]);
    }

    return 0;
}

// function to sort the stack
void sort_stack(int stack[], int size) {
    bubble_sort(stack, size);
}
```

```
// function to perform bubble sort
void bubble_sort(int arr[], int size) {
    int i, j, temp;

    for(i=0; i<size-1; i++) {
        for(j=0; j<size-i-1; j++) {
            if(arr[j] > arr[j+1]) {
                // swap the elements
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

#### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the size of the stack: 5
Enter the elements of the stack: 2 4 6 8 3
Sorted stack: 2 3 4 6 8
```

**Q. Program for implementation of Multiple stack in one Array.**

**ALGORITHM Minimum()**

**BEGIN :**

**BEGIN:**

1. Create a array.
2. Divide it into 2 parts.
3. Maxsize of one will be the start of another.
4. Perform all primitive operations by iterative method

**END;**

**Time Complexity:  $\Theta(1)$**

**Space Complexity:  $\Theta(1)$**

**CODE**

```
# define maxsize 100
```

```
int a[maxsize];
```

```
int top1=-1;
```

```
int top2=69;
```

```
int top3=maxsize;
```

```
int push1(int x)
```

```
{
```

```
    if(top1==top2-1)
```

```
    {
```

```
        return -1;
```

```
    }
```

```
    top1++;
```

```
    a[top1]=x;
```

```
    return 1;
```

```
}
```

```
int push2(int x)
```

```
{
```

```
    if(top2==top1+1)
```

```
    {
```

```
        return -1;
```

```
    }
```

```
    top2--;
```

```
    a[top2]=x;
```

```
    return 1;
```

```
}
```

```
int push3(int x)
```

```
{
```

```
    if(top3==70)
```

```
    {
```

```
        return -1;
```

```
    }
```

```
    top3--;
```

```
    a[top3]=x;
```



```

    return 1;
}

int pop1()
{
    int x;
    if(top1== -1)
    {
        return -1;
    }
    x=a[top1];
    top1--;
    return x;
}

int pop2()
{
    int x;
    if(top2==70)
    {
        return -1;
    }
    x=a[top2];
    top2++;
    return x;
}

int pop3()
{
    int x;
    if(top3==maxsize)
    {
        return -1;
    }
    x=a[top3];
    top3++;
    return x;
}

int peek1()
{
    if(top1== -1)
    {
        return -1;
    }
    return a[top1];
}

```

```
int peek2()
{
    if(top2==69)
    {
        return -1;
    }
    return a[top2];
}
```

```
int peek3()
{
    if(top3==100)
    {
        return -1;
    }
    return a[top3];
}
```

```
void display1()
{
    if(top1== -1)
    {
        return ;
    }
    printf("%d",a[top1]);
    top1--;
    display1();
}
```

```
void display2()
{
    if(top2==69)
    {
        return ;
    }
    printf("%d", a[top2]);
    top2++;
    display2();
}
```

```
void display3()
{
    if(top3==maxsize)
    {
        return ;
    }
    printf("%d", a[top3]);
    top3++;
    display3();
}
```

```

}
int main()
{ cout<<"Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024 \n";
  push1(1);
  push1(2);
  push1(3);
  pop1();
  cout<<peek1();
  display1();
  cout<<endl;
  push2(1);
  push2(2);
  push2(3);
  pop2();
  peek2();
  display2();

  push3(1);
  push3(2);
  push3(3);
  pop3();
  peek3();
  display3();
return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
3
2
2 1
3
2
2 1
3
2
2 1

```

---

### Experiment. 54

**OBJECT:** Program for Array implementation of Linear Queue

• **ALGORITHM INITIALIZE(Queue Q)**

```
BEGIN:
    Q.REAR=0
    Q.FRONT=1
END;
```

• **ALGORITHM ENQUEUE(Queue Q, key)**

```
BEGIN:
IF Q.REAR == SIZE THEN
    write (Queue overflow)
    Exit 1
Q.REAR=Q.REAR+1
Q.item[Q.REAR]=key
END;
```

• **ALGORITHM DEQUEUE(Queue Q)**

```
BEGIN:
IF Q.REAR-Q.FRONT+1==0 THEN
    Exit(1)
x=Q.item[Q.FRONT]
Q.FRONT =Q.FRONT +1
RETURN x
END;
```

• **ALGORITHM EMPTY(Queue Q)**

```
BEGIN:
    IF Q.REAR – Q.FRONT +1 == 0 THEN
        RETURN TRUE
    ELSE
        RETURN FALSE
END;
```

For all the above ALGORITHM

**Time Complexity:  $\Theta(1)$**

**Space Complexity:  $\Theta(1)$**

**CODE**

```
#include<stdio.h>
# define maxsize 100
struct queue
{
    int front,rear;
    int data[maxsize];
```

```

}q;

void initialize()
{
    q.front=q.rear=-1;
}

int isempty()
{
    if(q.front==-1 || q.front>q.rear)
    {
        return 1;
    }
    return 0;
}

int isfull()
{
    if(q.rear==maxsize-1)
    {
        return 1;
    }
    return 0;
}

void enqueue(int x)
{
    if(isfull())
    {
        return;
    }
    if(isempty())
    {
        q.front=q.rear=0;
    }
    else
    {
        q.rear++;
    }
    q.data[q.rear]=x;
}

int dequeue()
{
    int x;

```

```

    if(isempty())
    {
        return -1;
    }
    x=q.data[q.front];
    q.front++;
    return x;
}

```

```

int peek()
{
    if(isempty())
    {
        return -1;
    }
    int x;
    x=q.data[q.front];
    return x;
}

```

```

void show()
{
    if(isempty())
    {
        return ;
    }
    for(int i=q.front;i<=q.rear;i++)
    {
        Printf("%d",q.data[i]<<" ");
    }
}

```

```

int main()
{
    Printf("Name-Anubhav Singhal \\\ CS-A \\\ 2100320120024");
    initialize();
    enqueue(2);
    enqueue(3);
    enqueue(5);
    enqueue(4);
    enqueue(8);
    dequeue();
}

```

```
    show();  
return 0;  
}
```

#### **Output**

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
2  
3 5 4 8  
PS C:\Users\91807\Documents\DS Lab> █
```

## OBJECT: Program for Array implementation of Circular Queue

### •ALGORITHM INTIALIZATION(Queue Q)

```
BEGIN :  
    CQ.REAR=Size-1  
    CQ.FRONT=Size-1  
END;
```

### •ALGORITHM ENQUEUE(CQUEUE CQ,KEY)

```
BEGIN:  
    IF (CQ.REAR+1)%SIZE==CQ.FRONT THEN  
        write(Queue overflows)  
        Exit(1)  
    CQ.REAR=(CQ.REAR+1)%SIZE  
    CQ.item[CQ.REAR]=key  
END;
```

### •ALGORITHM DEQUEUE(CQUEUE CQ)

```
BEGIN:  
    IF CQ.REAR==CQ.FRONT  
        write( queue overflow)  
        Exit(1)  
    CQ.FRONT=(CQ.FRONT+1)%Size  
    x=CQ.item[CQ.FRONT]  
    RETURN(x)  
END;
```

### •ALGORITHM EMPTY(CQUEUE CQ)

```
BEGIN:  
    IF CQ.REAR = CQ.FRONT THEN  
        RETURN TRUE  
    ELSE  
        RETURN FALSE  
END;
```

For all the above ALGORITHM

**Time Complexity:  $\Theta(1)$**

**Space Complexity:  $\Theta(1)$**

### CODE

```
#include<stdio.h>  
# define maxsize 5  
struct queue  
{  
    int front,rear;  
    int data[maxsize];  
}q;
```



```

void initialize()
{
    q.front=q.rear=-1;
}

int isempty()
{
    if(q.front== -1)
    {
        return 1;
    }
    return 0;
}

int isfull()
{
    if((q.rear+1)%maxsize==q.front)
    {
        return 1;
    }
    return 0;
}

void enqueue(int x)
{
    if(isfull())
    {
        cout<<"Overflow";
        return;
    }
    if(isempty())
    {
        q.front=q.rear=0;
    }
    else
    {
        q.rear=(q.rear+1)%maxsize;
    }
    q.data[q.rear]=x;
    cout<<"item pushed successfully "<<endl;
}

int dequeue()
{
    int x;
    if(isempty())
    {
        cout<<"underflow";
    }
}

```

```

        return -1;
    }
    x=q.data[q.front];
    if(q.front==q.rear)
    {
        initialize();
        return x;
    }
    q.front=(q.front+1)%maxsize;
    return x;
}

```

```

int peek()
{
    if(isempty())
    {
        cout<<"Underflow";
        return -1;
    }
    int x;
    x=q.data[q.front];
    return x;
}

```

```

void show()
{
    if(isempty())
    {
        cout<<"Underflow";
        return ;
    }
    for(int i=q.front;i!=q.rear;i=(i+1)%maxsize)
    {
        Printf( "%d", q.data[i]);
    }
    Printf( "%d", q.data[q.rear]);
}

```

```

int main()
{
    Printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024");
    initialize();
    enqueue(2);
    enqueue(3);
    enqueue(5);
}

```

```
    enqueue(4);  
    enqueue(8);  
    dequeue();  
    show();  
return 0;  
}
```

### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
2  
3 5 4 8  
PS C:\Users\91807\Documents\DS Lab> █
```

### Experiment.56

**OBJECT:** Program for Array implementation of Double Ended Queue

•**ALGORITHM INSREAR (DQUEUE DQ)**

```
BEGIN:

    IF(DQ.REAR==SIZE-1) THEN
        Write(queue overflow)
        Exit(1)

    ELSE
        REAR=REAR+1
        DQ.REAR=item

    IF(REAR=0) THEN:
        REAR=REAR+1

    IF(FRONT=0) THEN:
        FRONT= FRONT +1

END;
```

•**ALGORITHM INSFRONT(DQUEUE DQ)**

```
BEGIN:

    IF(FRONT<=1) THEN:
        Write(cannot add item at FRONT end)
        Exit(1)

    ELSE
        FRONT=FRONT-1;
        DQ.FRONT=item

END;
```

•**ALGORITHM DELFRONT(DQUEUE DQ)**

```
BEGIN:

    IF(FRONT=0) THEN:
        Write(queue underflow)
        Exit(1)

    ELSE
        Item=DQ.FRONT
        Write(item)
```

```

        IF(FRONT=REAR) THEN:
            FRONT=0
            REAR=0
        ELSE
            FRONT=FRONT+1
    RETURN item
END;

```

• **ALGORITHM DELREAR(DQUEUE DQ)**

```

BEGIN:

    IF(REAR=0) THEN:
        Write(cannot delete value at REAR end)
        Exit(1)
    ELSE
        Item=DQ.REAR
        Write(item)

        IF(FRONT=REAR) THEN:
            FRONT=0
            REAR=0
        ELSE
            REAR=REAR-1
    RETURN item

END;

```

For all the above ALGORITHM

**Time Complexity:  $\Theta(1)$**

**Space Complexity:  $\Theta(1)$**

**CODE**

```

#include<stdio.h>
# define maxsize 100
struct queue
{
    int data[maxsize];
    int front,rear;
}q;

void initialize()
{
    q.front=q.rear=-1;
}
int emptymess()
{
    return q.front== -1;
}

```

```

int fullness()
{
    return (q.rear+1)%maxsize==q.front;
}

```

```

void enqueue_at_rear(int x)
{
    if(fullness())
    {
        cout<<"Overflow";
        return;
    }

```

```

    if(emptymess())
    {
        q.front=q.rear=0;
    }
    else
    {
        q.rear=(q.rear+1)%maxsize;
    }
    q.data[q.rear]=x;
}

```

```

int dequeue_at_rear()
{
    if(emptymess())
    {
        return -1;
    }

```

```

    int x=q.data[q.rear];
    if(q.rear==q.front)
    {
        initialize();
        return x;
    }
    q.rear=(q.rear-1+maxsize)%maxsize;    // because 0-1 hone se galat ho jayega par +maxsize
    se wo sabse end me(maxsize-1) par chala jayega
    return x;
}

```

```

int dequeue_at_front()
{
    if(emptymess())
    {
        cout<<"Underflow";
    }

    int x=q.data[q.front];
    if(q.rear==q.front)
    {
        initialize();
        return x;
    }
    q.front=(q.front+1)%maxsize;
    return x;
}

void enqueue_at_front(int x)
{
    if(fullness())
    {
        cout<<"Overflow";
        return ;
    }

    if(emptymess())
    {
        q.front=q.rear=0;
    }
    else
    {
        q.front=(q.front-1+maxsize)%maxsize;
    }
    q.data[q.front]=x;
}

int main()
{
    Printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024");
    initialize();
    Enqueue_at_front(2);
    Enqueue_at_rear(3);
    Enqueue_at_rear (5);
    Enqueue_at_rear (4);
    Enqueue_at_rear (8);
}

```

```
    Dequeue_at_rear();  
    show();  
return 0;  
}
```

### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
2  
3 5 4 8  
PS C:\Users\91807\Documents\DS Lab> █
```

### Experiment. 57,58

**OBJECT:** Program for Array implementation of priority Queue

• **ALGORITHM** ARRAY INSERTION(pq[],i,key,:\*N)



```

BEGIN:
    For j=*N-1 to i STEP-1 Do
        pq[j+1]=pq[j]
    pq[i]=key
    *N=*N+1
END;

```

TIME COMPLEXITY:  $\Omega(1)$ ,  $O(N)$   
 SPACE COMPLEXITY:  $\Theta(1)$

•**ALGORITHM ARRAYDELETE**(pq[], \*N, i)

```

BEGIN: y=pq[i]
    For j=i+1 to *N-1 Do
        pq[j-1]=pq[j]
    *N=*N-1
    RETURN y
END;

```

TIME COMPLEXITY:  $\Omega(1)$ ,  $O(N)$   
 SPACE COMPLEXITY:  $\Theta(1)$

•**ALGORITHM ARRAYINSERTION**(pq[], \*N, key)

```

BEGIN:
    i=0
    WHILE i<=*N && key>pq[i]
        i=i+1
    Arrayinsertion(pq,i,key,N)
END;

```

•**ALGORITHM REMOVE** (pq[], \*N)

```

BEGIN:
    x=Arraydelete(pq,N,1)
    RETURN x
END;

```

## Ascending

```

#include<iostream>
using namespace std;
# define maxsize 100;
struct item
{
    int value,priority;
};

```

```
struct queue
{
    int front,rear;
    struct item data[maxsize];
}q;
```

```
void initialize()
{
    q.front=q.rear=-1;
}
```

```
int emptyness()
{
    if(q.front==-1)
    {
        return 1;
    }
    return 0;
}
```

```
int fullness()
{
    return (q.rear+1)%maxsize==q.front;
}
```

```
void enqueue(struct item x)
{
    if(fullness())
    {
        return;
    }
}
```

```
if(emptyness())
{
    q.front=q.rear=0;
    q.data[q.rear]=x;
    return;
}
q.rear=(q.rear+1)%maxsize;
q.data[q.rear]=x;
int i=q.rear;
int j=(i-1+maxsize)%maxsize;
```

```

while(q.data[i].priority<q.data[j].priority)
{
    swap(q.data[i],q.data[j]);
    if(j==q.front)
    {
        return;
    }
    i=(i-1+maxsize)%maxsize;
    j=(i-1+maxsize)%maxsize;
}
return;
}

```

```

struct item dequeue()
{
    struct item x;
    if(emptyness())
    {
        x.priority=x.value=-1;
        return x;
    }
    else
    {
        x=q.data[q.front];
        if(q.front==q.rear)
        {
            struct item y=x;
            initialize();
            return y;
        }
        q.front=(q.front+1)%maxsize;
        return x;
    }
}

```

```

void show()
{
    if(emptyness())
    {
        return ;
    }
    for(int i=q.front;i!=q.rear;i=(i+1)%maxsize)
    {

```

```

    Printf("%d",cout<<q.data[i].value);
}
Printf("%d",cout<<q.data[i].value);
}

int main()
{
printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");

    struct item x;
x.value=9,x.priority=3;
enqueue(x);
x.value=91,x.priority=2;
enqueue(x);
x.value=1,x.priority=1;
enqueue(x);
x.value=2,x.priority=6;
enqueue(x);
dequeue();
show();
return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
1 91 9 2

```

### Descending

```

#include<stdio.h>
# define maxsize 100
struct item
{
    int value,priority;
};

struct queue
{
    int front,rear;
    struct item data[maxsize];
}q;

```

```
void initialize()
{
    q.front=q.rear=-1;
}
```

```
int emptyness()
{
    if(q.front== -1)
    {
        return 1;
    }
    return 0;
}
```

```
int fullness()
{
    return (q.rear+1)%maxsize==q.front;
}
```

```
void enqueue(struct item x)
{
    if(fullness())
    {
        cout<<"Overflow";
    }
}
```

```
if(emptyness())
{
    q.front=q.rear=0;
    q.data[q.front]=x;
    return;
    q.rear=(q.rear+1)%maxsize;
    q.data[q.rear]=x;
}
```

```
int i=q.rear;
int j=(i-1+maxsize)%maxsize;
```

```
while(q.data[i].priority>q.data[j].priority)
{
    swap(q.data[i],q.data[j]);
    if(j==q.front)
    {
        return;
    }
    i=(i-1+maxsize)%maxsize;
```

```

    j=(i-1+maxsize)%maxsize;
}
return;
}

```

```

struct item dequeue()
{

```

```

    struct item x;
    if(emptyness())
    {
        x.priority=x.value=-1;
        return x;
    }

```

```

    x.value=q.data[q.rear].value;
    x.priority=q.data[q.rear].priority;
    if(q.front==q.rear)
    {
        initialize();
        return x;
    }
    q.rear=(q.rear-1+maxsize)%maxsize;
    return x;
}

```

```

void show()
{

```

```

    if(emptyness())
    {
        cout<<"Underflow";
        return ;
    }

```

```

    for(int i=q.front;i!=q.rear;i=(i+1)%maxsize)
    {
        cout<<q.data[i].value<<" ";
    }
    cout<<q.data[q.rear].value;
}

```

```

int main()
{

```

```

    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024 \n");
    struct item x;
    x.value=9,x.priority=3;

```

```

enqueue(x);
x.value=91,x.priority=2;
enqueue(x);
x.value=1,x.priority=1;
enqueue(x);
x.value=2,x.priority=6;
enqueue(x);
dequeue();
show();
return 0;
}

```

```

return 0;
}

```

#### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
2 9 91 1

```

#### Experiment. 59

**OBJECT:** Program for Stack implementation using Queue.

#### •ALGORITHM

BEGIN:

1. All operations are same as in dequeue.
2. Insertion and deletion take place at the same end.

END;

TIME COMPLEXITY:  $\Omega(1)$ ,  $O(1)$

SPACE COMPLEXITY:  $\Theta(1)$

### CODE

```
#include<stdio.h>

# define maxsize 100
struct queue
{
    int front,rear;
    int data[maxsize];
}q;

void initialize()
{
    q.front=q.rear=-1;
}

int isempty()
{
    if(q.front== -1 || q.front>q.rear)
    {
        return 1;
    }
    return 0;
}

int isfull()
{
    if(q.rear==maxsize-1)
    {
        return 1;
    }
    return 0;
}

void push(int x)
{
    if(isfull())
    {
        return;
    }
    if(isempty())
    {
        q.front=q.rear=0;
    }
    else
    {
        q.rear++;
    }
}
```



```

    }
    q.data[q.rear]=x;
}

int pop()
{
    int x;
    if(isempty())
    {
        return -1;
    }
    x=q.data[q.rear];
    q.rear--;
    return x;
}

int peek()
{
    if(isempty())
    {
        return -1;
    }
    int x;
    x=q.data[q.rear];
    return x;
}

void show()
{
    if(isempty())
    {
        return ;
    }

    for(int i=q.rear;i>=q.front;i--)
    {
        cout<<q.data[i]<<" ";
    }
    initialize();
}

int main()
{
    int choice,x;

    while(1)
    {
        printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    }
}

```

```
printf("enter choice \n 1.push1 \n 2.push2 \n 3.pop1 \n 4.pop2 \n 5.peek1 \n 6.peek2 \n 7.display1 \n 8.display2 \n 9.exit");  
printf(" enter choice ");  
scanf("%d",& choice);
```

```
switch(choice)  
{  
    case 1:  
    {  
        printf(" enter item ");  
scanf("%d",&x);  
        int i=push1(x);  
        if(i==-1)  
            printf(" k overflow");  
        else  
            printf(" item pushed successfully ");  
        break;  
    }  
}
```

```
case 2:  
{  
    printf(" enter item ");  
    scanf("%d",&x);  
    int i=push2(x);  
    if(i==-1)  
        printf(" k overflow");  
    else  
        printf("item pushed successfully ");  
    break;  
}
```

```
case 3:  
{  
    x=pop1();  
    if(x==-1)  
        printf(" underflow");  
    else  
        printf(" oped item is %d"x);  
    break;  
}
```

```
case 4:  
{  
    x=pop2();  
    if(x==-1)  
        printf(" underflow");  
    else  
        printf(" popped item is %d"x);  
    break;  
}
```

```

case 5:
{
    int x=peek1();
    if(x==-1)
    {
        printf(" Stack underflow");
        break;
    }
    printf(" topmost element of stack is %d",x);
    break;
}

case 6:
{
    int x=peek2();
    if(x==-1)
    {
        printf(" Stack underflow");
        break;
    }
    printf(" topmost element of stack is %d ",x);
    break;
}

case 7:
{
    printf(" stack is :");
    display1();
    break;
}
case 8:
{
    printf(" stack is :");
    display2();
    break;
}
case 9:
    exit(0);
    break;
default:
    printf(" incorrect choice");
    break;
}
}
return 0;
}

```

## Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
enter choice
1.push1
2.push2
3.pop1
4.pop2
5.peek1
6.peek2
7.display1
8.display2
9.exit
enter choice 1
enter item 2
item pushed successfully

enter choice 1
enter item 4
item pushed successfully

enter choice 3
poped item is 4

enter choice 5
topmost element of stack is 2

enter choice 7
stack is :2

enter choice 2
enter item 4
item pushed successfully

enter choice 4
poped item is 4

enter choice 7
stack is :

```

### Experiment. 60

**OBJECT:** Program for Queue implementation using Stack.

**•ALGORITHM**

BEGIN:

1. All the operations are performed using 2 stacks.
2. Item is enqueued at the top.
3. For peek and pop all elements are popped and pushed into other stack and process reversed later.

END;

TIME COMPLEXITY:  $\Omega(1)$ ,  $O(n)$   
SPACE COMPLEXITY:  $\Theta(n)$

## CODE

```
#include<stdio.h>
# define maxsize 100
struct stack
{
    int top;
    int data[maxsize];
};

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

int emptyness(struct stack *s)
{
    if(s->top==-1)
        return 1;
    return 0;
}

int fullness(struct stack *s)
{
    if(s->top==maxsize-1)
    {
        return 1;
    }
    return 0;
}

int enqueue(struct stack *s,int x)           // push
{
    if(fullness(s))
    {
        return -1;
    }
    s->top++;
    s->data[s->top]=x;
    return 1;
}

int pop(struct stack *s)
{

```

```

int x;
if(emptyness(s))
{
    return -1;
}
x=s->data[s->top];
s->top--;
return x;
}

```

```

int dequeue(struct stack *s)
{
    if(emptyness(s))
    {
        return -1;
    }
    struct stack t;
    initialize(&t);
    while(!emptyness(s))
    {
        enqueue(&t,pop(s));
    }
    int x=pop(&t);
    while(!emptyness(&t))
    {
        enqueue(s,pop(&t));
    }
    return x;
}

```

```

int peek(struct stack *s)
{
    if(emptyness(s))
    {
        return -1;
    }
    struct stack t;
    struct stack *z=&t;
    initialize(&t);
    while(!emptyness(s))
    {
        enqueue(&t,pop(s));
    }
    int x=z->data[z->top];
    while(!emptyness(z))
    {
        enqueue(s,pop(z));
    }
}

```

```

    }
    return x;
}

void display(struct stack *s)
{
    if(emptyness(s))
    {
        return ;
    }
    cout<<s->data[s->top]<<" ";
    s->top--;
    display(s);
}

int main()
{
    Printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024");
    initialize();
    enqueue(2);
    enqueue(3);
    enqueue(5);
    enqueue(4);
    enqueue(8);
    dequeue();
    show();
return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
2
3 5 4 8
PS C:\Users\91807\Documents\DS Lab>

```

### Experiment 61

**OBJECT-Program for Linear Linked List primitive operations**

**DOMAIN-Linked List**

**ALGORITHM:**

<p><b>ALGORITHM:</b></p> <p><b>BEGIN:</b></p>	<p><b>InsertBeginning(START,key)</b></p> <p>p=getnode()</p> <p>p-&gt;info=key</p> <p>p-&gt;next=START</p> <p>START=p</p>
---	--

**END;**  
TIME COMPLEXITY:  $\Theta(1)$   
SPACE COMPLEXITY:  $\Theta(1)$

**ALGORITHM:** **InsertAfter(p, key)**  
**BEGIN:**  
q=getnode()  
q->info=key  
q->next=p->next  
p->next=q

**END;**  
TIME COMPLEXITY:  $\Theta(1)$   
SPACE COMPLEXITY:  $\Theta(1)$

**ALGORITHM:** **InsertEnd(START, key)**  
**BEGIN:**  
IF START==NULL THEN  
    InsertBeginning (START, key) p=START  
  
ELSE  
    P=START  
    WHILE(p!=NULL) DO  
        q=getnode()  
        q->info=key  
        q->next=NULL  
        p->next=q

**END;**  
TIME COMPLEXITY:  $\Theta(N)$   
SPACE COMPLEXITY:  $\Theta(1)$

**ALGORITHM:** **DelBeg(START)**  
**BEGIN:**  
IF START==NULL THEN  
    Write("void deletion")  
    Exit(1)  
ELSE  
    p=START  
    START=p->next  
    x=p->info  
    RETURN x

**END;**  
TIME COMPLEXITY:  $\Theta(1)$   
SPACE COMPLEXITY:  $\Theta(1)$

**ALGORITHM:** **DelEnd(START)**  
**BEGIN:**  
p=START  
q=NULL  
WHILE(next(p)!=NULL)  
    q=p



```

        p=p->next
        q->next=NULL
        x=p->info
        freenode(p)
        RETURN x

```

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SAPCE COMPLEXITY:  $\Theta(1)$

### CODE

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

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

```
struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
};
```

```
struct Node* head = NULL;
```

```
// Function to insert a new node at the beginning of the linked list
```

```
void insertAtBeginning(int value) {
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    newNode->data = value;
```

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

```
    head = newNode;
```

```
}
```

```
// Function to insert a new node at the end of the linked list
```

```
void insertAtEnd(int value) {
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    newNode->data = value;
```

```
    newNode->next = NULL;
```

```
    if (head == NULL) {
```

```
        head = newNode;
```

```
        return;
```

```
    }
```

```
    struct Node* lastNode = head;
```

```
    while (lastNode->next != NULL) {
```

```
        lastNode = lastNode->next;
```

```
    }
```

```
    lastNode->next = newNode;
```

```
}
```

```
// Function to insert a new node after a given node
```

```
void insertAfterNode(struct Node* prevNode, int value) {
```

```
    if (prevNode == NULL) {
```

```

        printf("Previous node cannot be NULL");
        return;
    }
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = prevNode->next;
    prevNode->next = newNode;
}

```

// Function to delete a node with a given value from the linked list

```

void deleteNode(int value) {
    struct Node* currentNode = head;
    struct Node* prevNode = NULL;
    if (currentNode != NULL && currentNode->data == value) {
        head = currentNode->next;
        free(currentNode);
        return;
    }
    while (currentNode != NULL && currentNode->data != value) {
        prevNode = currentNode;
        currentNode = currentNode->next;
    }
    if (currentNode == NULL) {
        return;
    }
    prevNode->next = currentNode->next;
    free(currentNode);
}

```

// Function to print the linked list

```

void printList() {
    struct Node* currentNode = head;
    while (currentNode != NULL) {
        printf("%d ", currentNode->data);
        currentNode = currentNode->next;
    }
}

```

```

int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    insertAtEnd(5);
    insertAtBeginning(2);
    insertAtBeginning(1);
    insertAtEnd(6);
    insertAfterNode(head->next, 3);
}

```

```
printf("Original linked list: ");  
printList();  
deleteNode(3);  
deleteNode(1);  
printf("\nLinked list after deleting nodes with value 3 and 1: ");  
printList();  
return 0;  
}
```

### Output

Name-Anubhav Singhal \ CS-A \ 2100320120024

Original linked list: 1 2 3 5 6

Linked list after deleting nodes with value 3 and 1: 2 5 6

## Experiment 62

**OBJECT-** Program for creation of Linked List header file and test of basic functions through that  
**DOMAIN-** Linked List

**ALGORITHM:**

```
· ALGORITHM:      InsertBeginning(START,key)
  BEGIN:          p=getnode()
                   p->info=key
                   p->next=START
                   START=p

  END;
TIME COMPLEXITY:  $\Theta(1)$ 
SPACE COMPLEXITY:  $\Theta(1)$ 
```

```
· ALGORITHM:      InsertAfter(p,key)
  BEGIN:          q=getnode()
                   q->info=key
                   q->next=p->next
                   p->next=q

  END;
TIME COMPLEXITY:  $\Theta(1)$ 
SPACE COMPLEXITY:  $\Theta(1)$ 
```

```
· ALGORITHM:      InsertEnd(START,key)
  BEGIN:          IF START==NULL THEN
                     InsertBeginning (START,key) p=START

  ELSE
    P=START
    WHILE(p!=NULL) DO
      q=getnode()
      q->info=key
      q->next=NULL
      p->next=q

  END;
TIME COMPLEXITY:  $\Theta(N)$ 
SPACE COMPLEXITY:  $\Theta(1)$ 
```

```
· ALGORITHM:      DelBeg(START)
  BEGIN:          IF START==NULL THEN
                     Write("void deletion")
                     Exit(1)
  ELSE
    p=START
    START=p->next
    x=p->info
    RETURN x

  END;
```

TIME COMPLEXITY:  $\Theta(1)$   
SPACE COMPLEXITY:  $\Theta(1)$

**ALGORITHM:** **DelEnd(START)**  
**BEGIN:**  
    p=START  
    q=NULL  
    WHILE(next(p)!=NULL)  
        q=p  
        p=p->next  
    q->next=NULL  
    x=p->info  
    freenode(p)  
    RETURN x  
**END;**

TIME COMPLEXITY:  $\Theta(1)$   
SPACE COMPLEXITY:  $\Theta(1)$

## CODE

### Linked list.h

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

// Function to insert a new node at the beginning of the linked list
void insertAtBeginning(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}

// Function to insert a new node at the end of the linked list
void insertAtEnd(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
        return;
    }
    struct Node* lastNode = head;
```

```

while (lastNode->next != NULL) {
    lastNode = lastNode->next;
}
lastNode->next = newNode;
}

```

```

// Function to insert a new node after a given node
void insertAfterNode(struct Node* prevNode, int value) {
    if (prevNode == NULL) {
        printf("Previous node cannot be NULL");
        return;
    }
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = prevNode->next;
    prevNode->next = newNode;
}

```

```

// Function to delete a node with a given value from the linked list
void deleteNode(int value) {
    struct Node* currentNode = head;
    struct Node* prevNode = NULL;
    if (currentNode != NULL && currentNode->data == value) {
        head = currentNode->next;
        free(currentNode);
        return;
    }
    while (currentNode != NULL && currentNode->data != value) {
        prevNode = currentNode;
        currentNode = currentNode->next;
    }
    if (currentNode == NULL) {
        return;
    }
    prevNode->next = currentNode->next;
    free(currentNode);
}

```

```

// Function to print the linked list
void printList() {
    struct Node* currentNode = head;
    while (currentNode != NULL) {
        printf("%d ", currentNode->data);
        currentNode = currentNode->next;
    }
}

```

```
}
```

### Implementation

```
#include <stdio.h>
#include <stdlib.h>
#include "linked list.h"
int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \\n");
    insertAtEnd(5);
    insertAtBeginning(2);
    insertAtBeginning(1);
    insertAtEnd(6);
    insertAfterNode(head->next, 3);
    printf("Original linked list: ");
    printList();
    deleteNode(3);
    deleteNode(1);
    printf("\\nLinked list after deleting nodes with value 3 and 1: ");
    printList();
    return 0;
}
```

### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Original linked list: 1 2 3 5 6
Linked list after deleting nodes with value 3 and 1: 2 5 6
```

### Experiment 63

**OBJECT-** Program for finding count of Nodes in Linked List

**DOMAIN-** Linked List

**ALGORITHM:****ALGORITHM****BEGIN:****nodes(START)**

```
p=START
Count=0
WHILE(p!=NULL) DO
    count++
    p=p->next->next
```

**END;**TIME COMPLEXITY:  $\Theta(N)$ SPACE COMPLEXITY:  $\Theta(1)$ **CODE**

#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

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

```
int countNodes(struct Node* head) {
    int count = 0;
    struct Node* current = head;
    while (current != NULL) {
        count++;
        current = current->next;
    }
    return count;
}
```

```
int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;

    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 1;
    head->next = second;

    second->data = 2;
    second->next = third;

    third->data = 3;
```



```
third->next = NULL;

int count = countNodes(head);
printf("Number of nodes in the linked list: %d\n", count);
return 0;
}
```

#### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Number of nodes in the linked list: 3
```

#### Experiment 64

**OBJECT-** Program for concatenation of linear linked list.

**DOMAIN-** Linked List

**ALGORITHM**                      Concatenate(START1, START2)

```

BEGIN:                                IF START1==null THEN
                                      RETURN (START2)
                                      ELSE
                                          IF START2==null THEN
                                              RETURN (START1)
                                          ELSE
                                              p=START1
                                              WHILE (p->next != null) do
                                                  p=p->next
                                              p->next=START2
                                      RETURN (START1)
END;

```

TIME COMPLEXITY:  $\Theta(N)$ : N is the length of Linked List 1  
SPACE COMPLEXITY:  $\Theta(1)$

#### CODE

```

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

void printList(struct Node *head) {
    struct Node *temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void push(struct Node **head_ref, int new_data) {
    struct Node *new_node = (struct Node*) malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void concatenate(struct Node **head1, struct Node **head2) {
    if (*head1 == NULL) {
        *head1 = *head2;
        return;
    }
    if (*head2 == NULL) {

```

```

        return;
    }
    struct Node *temp = *head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = *head2;
}
int main() {
    struct Node *head1 = NULL;
    struct Node *head2 = NULL;
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024 \n");
    push(&head1, 3);
    push(&head1, 2);
    push(&head1, 1);

    push(&head2, 6);
    push(&head2, 5);
    push(&head2, 4);

    printf("List 1: ");
    printList(head1);
    printf("List 2: ");
    printList(head2);

    concatenate(&head1, &head2);
    printf("Concatenated List: ");
    printList(head1);

    return 0;
}

```

#### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
List 1: 1 2 3
List 2: 4 5 6
Concatenated List: 1 2 3 4 5 6

```

#### Program 65

Program to implement Linear search.

#### Algorithm

1. Begin
2. Create a function to perform linear search in a linked list.

3. The function should take two arguments: a pointer to the head of the linked list and the value to be searched.
4. Traverse the linked list from the head node to the last node.
5. Check if the current node's data matches the search value.
6. If it does, return the current node's index (or position) in the linked list.
7. If it does not, move to the next node.
8. If the end of the linked list is reached and the value is not found, return -1 (or some other value to indicate that the search was unsuccessful).
9. END

## CODE

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

void push(struct Node **head_ref, int new_data) {
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

int search(struct Node *head, int key) {
    struct Node *current = head;
    int position = 1;
    while (current != NULL) {
        if (current->data == key) {
            return position;
        }
        position++;
        current = current->next;
    }
    return -1;
}

int main() {
    struct Node *head = NULL;
    int key, position;
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");

    push(&head, 9);
    push(&head, 7);
```

```

push(&head, 5);
push(&head, 3);
push(&head, 1);

printf("Enter a key to search in the linked list: ");
scanf("%d", &key);
position = search(head, key);

if (position == -1) {
    printf("Key not found in the linked list.\n");
} else {
    printf("Key found at position %d in the linked list.\n", position);
}
return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter a key to search in the linked list: 3
Key found at position 2 in the linked list.

```

### Program 66

**Program to insert an item at any given position in the linked List.**

#### Algorithm

1. Create a new node with the given data.
2. If the position is 0, set the new node as the head and update the size of the linked list.
3. Traverse the linked list till the position before the given position.

4. If the position is greater than the size of the linked list, return an error message.
5. Get the next node of the current node and assign it to the next node of the new node.
6. Assign the new node to the next node of the current node.
7. Update the next pointer of the new node.
8. Increment the size of the linked list.
9. Return the updated linked list.

#### CODE

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

void insertAtPos(struct node **head_ref, int data, int pos){
    struct node *new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = data;
    new_node->next = NULL;

    struct node *temp = *head_ref;
    if(pos == 1){
        new_node->next = *head_ref;
        *head_ref = new_node;
        return;
    }
    for(int i=1; i<pos-1 && temp!=NULL; i++){
        temp = temp->next;
    }
    if(temp == NULL){
        printf("Invalid position\n");
        return;
    }
    new_node->next = temp->next;
    temp->next = new_node;
}

void displayList(struct node *head){
    while(head != NULL){
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
```

```

int main(){
    struct node *head = NULL;
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    insertAtPos(&head, 10, 1); // inserting at beginning
    insertAtPos(&head, 20, 2); // inserting at position 2
    insertAtPos(&head, 30, 3); // inserting at position 3
    insertAtPos(&head, 40, 4); // inserting at position 4
    insertAtPos(&head, 50, 5); // inserting at position 5

    printf("Linked List: ");
    displayList(head);
    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Linked List: 10 -> 20 -> 30 -> 40 -> 50 -> NULL

```

### Practical-67: creation a copy of linked list

**DOMAIN:** Linked List

**ALGORITHM:**

### ALGORITHM

#### BEGIN:

1. Create a new linked list with a dummy head node.
2. Traverse the original linked list and for each node, create a new node and copy the data from the original node to the new node.

3. Insert the new node at the end of the new linked list.
4. Return the head of the new linked list (i.e., the node after the dummy head).

**END;**

## CODE

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

struct Node* copyLinkedList(struct Node* head) {
    struct Node* newHead = (struct Node*)malloc(sizeof(struct Node));
    newHead->data = 0;
    newHead->next = NULL;
    struct Node* current = head;
    struct Node* newCurrent = newHead;
    while (current != NULL) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = current->data;
        newNode->next = NULL;

        newCurrent->next = newNode;
        newCurrent = newCurrent->next;

        current = current->next;
    }
    return newHead->next;
}

int main() {
    printf("Anubhav Roll.No: 2100320120024\n CS-A");
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 1;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 2;
    head->next->next = NULL;
    struct Node* newHead = copyLinkedList(head);
}
```



```

    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
    current = newHead;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");

    return 0;
}

```

### Output

c 1 2

### Program 68

**Program for counting nodes containing even and odd information.**

#### Algorithm

1. Create a variable to keep track of the number of nodes containing even information, initialize it to 0.
2. Create another variable to keep track of the number of nodes containing odd information, initialize it to 0.
3. Traverse the tree starting from the root node.
4. For each node, check if its information is even or odd.
5. If the information is even, increment the even count variable.
6. If the information is odd, increment the odd count variable.
7. Continue traversing the tree until all nodes have been visited.
8. Return the even and odd count variables.

## CODE

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

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

void addNode(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void countEvenOdd(Node* head, int* countEven, int* countOdd) {
    while (head != NULL) {
        if (head->data % 2 == 0) {
            (*countEven)++;
        } else {
            (*countOdd)++;
        }
        head = head->next;
    }
}
```

```
int main() {
    Node* head = NULL;
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024 \n");
    int countEven = 0;
    int countOdd = 0;

    addNode(&head, 1);
    addNode(&head, 2);
    addNode(&head, 3);
    addNode(&head, 4);
    addNode(&head, 5);

    countEvenOdd(head, &countEven, &countOdd);

    printf("Number of even nodes: %d\n", countEven);
    printf("Number of odd nodes: %d\n", countOdd);
    return 0;
}
```

#### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Number of even nodes: 2
Number of odd nodes: 3
```

### Experiment 70:

Object: Program for merging two unsorted Linked Lists

DOMAIN: Linked List

ALGORITHM:

ALGORITHM Union(START1, START2)

BEGIN:

```
    START3=NULL
    p=START 1
    q=START2
    WHILE(p!=NULL&&q!=NULL) DO
        IF(info(p)==info(q)) THEN
            p=next(p)
            q=next(q)
        ELSE
            IF(info(p)<info(q)) THEN
                insertend(START3,info(p))
                p=next(p)
            ELSE
                insertend(START3,info(q))
                q=next(q)
            ENDIF
        ENDIF
    ENDWHILE
    WHILE(p!=NULL) DO
        insertend(START3,info(p))
        p=next(p)
    ENDWHILE
    WHILE(q!=NULL) DO
        insertend(START3,info(q))
        q=next(q)
    ENDWHILE
```

```

        insertend(START3,info(q))
        q=next(q)
    RETURN START3
END;

```

**Time complexity:** $O(M+N)$

**Space complexity:** $O(M+N)$

#### CODE

```

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

// Function to insert a new node at the end of a linked list
void insertAtEnd(struct Node** headRef, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (*headRef == NULL) {
        *headRef = newNode;
    } else {
        struct Node* curr = *headRef;
        while (curr->next != NULL) {
            curr = curr->next;
        }
        curr->next = newNode;
    }
}

void printList(struct Node* head) {
    struct Node* curr = head;
    while (curr != NULL) {
        printf("%d ", curr->data);
        curr = curr->next;
    }
    printf("\n");
}

struct Node* mergeSortedLists(struct Node* head1, struct Node* head2) {
    if (head1 == NULL) {
        return head2;
    } else if (head2 == NULL) {
        return head1;
    } else {
        struct Node* result = NULL;

```

```

    if (head1->data <= head2->data) {
        result = head1;
        result->next = mergeSortedLists(head1->next, head2);
    } else {
        result = head2;
        result->next = mergeSortedLists(head1, head2->next);
    }
    return result;
}
}

```

```

int main() {
    // Example usage of the functions above
    struct Node* head1 = NULL;
    insertAtEnd(&head1, 1);
    insertAtEnd(&head1, 3);
    insertAtEnd(&head1, 5);
    insertAtEnd(&head1, 7);
    printf("List 1: ");
    printList(head1);

    struct Node* head2 = NULL;
    insertAtEnd(&head2, 2);
    insertAtEnd(&head2, 4);
    insertAtEnd(&head2, 6);
    printf("List 2: ");
    printList(head2);

    struct Node* mergedList = mergeSortedLists(head1, head2);
    printf("Merged List: ");
    printList(mergedList);

    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
List 1: 1 3 5 7
List 2: 2 4 6
Merged List: 1 2 3 4 5 6 7

```

### Experiment. 71

**Object:** *Program for finding difference of two linked list (consider lists as sets)*

**DOMAIN:** Linked List

**This finds  $A - B$  (A is the first Linked List and B the Second one)**

**ALGORITHM** Difference(list1,list2)

**BEGIN:**

```
list3=NULL
p=list1
q=list2
WHILE(p!=NULL&&q!=NULL) DO
    IF(info(p) == info(q)) THEN
        p=next(p)
        q=next(q)
    ELSE
        IF(info(p)<info(q)) THEN
            insend(list3,info(p))
            p=next(p)
        ELSE
            q=next(q)
        WHILE(p!=NULL) DO
            insertend(START3,info(p))
            p=next(p)
        RETURN list3
```

**END:**

**Time Complexity:** $O(M+N)$

**Space Complexity:** $O(N)$ : N is the length of first Linked List

#### **CODE**

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

void insert(struct Node** head, int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void printList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
    } else {
        printf("List: ");
        while (head != NULL) {
            printf("%d ", head->data);
            head = head->next;
        }
        printf("\n");
    }
}

struct Node* difference(struct Node* list1, struct Node* list2) {
    struct Node* result = NULL;
```



```

while (list1 != NULL) {
    int found = 0;
    struct Node* current = list2;

    while (current != NULL) {
        if (current->data == list1->data) {
            found = 1;
            break;
        }
        current = current->next;
    }

    if (!found) {
        insert(&result, list1->data);
    }

    list1 = list1->next;
}
return result;
}

int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    struct Node* result = NULL;

    insert(&list1, 1);
    insert(&list1, 2);
    insert(&list1, 3);
    insert(&list1, 4);
    insert(&list1, 5);

    insert(&list2, 3);
    insert(&list2, 4);
    insert(&list2, 5);
    insert(&list2, 6);
    insert(&list2, 7);

    // Print the two linked lists
    printList(list1);

```

```

printList(list2);

result = difference(list1, list2);
printf("Difference of the two linked lists:\n");
printList(result);
return 0;
}

```

#### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
List: 1 2 3 4 5
List: 3 4 5 6 7
Difference of the two linked lists:
List: 1 2

```

#### Experiment 114

**Object-**Write an ALGORITHM for creation of min heap and max heap.

**DOMAIN-**Heap

**ALGORITHM MaxHeapIFY(A[],N)**

**BEGIN:**

```

    FOR i=N/2 TO STEP-1 DO
        Adjust(A,i,N)

```

**END;**

**ALGORITHM Adjust(A[],i,N)**

**BEGIN:**

```

    WHILE 2*i<=N DO
        j=2*i
        IF j+1<=N THEN
            IF A[j+1]>A[j]
                j=j+1
        IF A[j]>A[i] THEN
            Exchange(A[j],A[i])
        ELSE
            BREAK
    i=j

```

**END;**

**ALGORITHM MinHeapIFY(A[],N)**

**BEGIN:**

```
    FOR i=N/2 TO STEP-1 DO  
        Adjust(A,i,N)
```

**END;**

**ALGORITHM Adjust(A[],i,N)**

**BEGIN:**

```
    WHILE 2*i<=N DO  
        j=2*i  
        IF j+1<=N THEN  
            IF A[j+1]<A[j]  
                j=j+1  
        IF A[j]<A[i] THEN  
            Exchange(A[j],A[i])  
        ELSE  
            BREAK  
    i=j
```

**END;**

**Time Complexity:  $\Theta(N\log N)$**

**Space Complexity:  $\Theta(1)$**

**CODE**

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

```
void swap(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
void createMinHeap(int arr[], int n) {  
    int i, j, k;  
    for (i = 1; i < n; i++) {  
        k = i;  
        j = (k - 1) / 2;  
        while (k > 0 && arr[k] < arr[j]) {  
            swap(&arr[k], &arr[j]);  
            k = j;  
            j = (k - 1) / 2;  
        }  
    }  
}
```

```
void createMaxHeap(int arr[], int n) {  
    int i, j, k;  
    for (i = 1; i < n; i++) {  
        k = i;
```

```

        j = (k - 1) / 2;
        while (k > 0 && arr[k] > arr[j]) {
            swap(&arr[k], &arr[j]);
            k = j;
            j = (k - 1) / 2;
        }
    }
}

int main() {
    int arr[] = {7, 6, 5, 4, 3, 2, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \\n");
    createMinHeap(arr, n);

    printf("Min heap: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\\n");

    createMaxHeap(arr, n);

    printf("Max heap: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}

```

## Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Min heap: 1 4 2 7 5 6 3
Max heap: 7 5 6 1 4 2 3

```

### Experiment 115

**Write an ALGORITHM for realizing Heap as Ascending/ Descending Priority Queue**

ALGORITHM PQInsert(A[],N,key)

BEGIN:

    A[N+1]=key

    i=N+1

    WHILE i>1 AND A[i]<A[i/2] DO

        Exchange(A[i],A[i/2])

        i=i/2

    N=N+1

END;

**Time Complexity:  $\Theta(\log N)$**

**Space Complexity:  $\Theta(1)$**

ALGORITHM PQDelete (A[],N)

BEGIN:

    x=A[i]

    A[i]=A[N]

    Adjust(A[],1,N-1)

    RETURN x

END;

**Time Complexity:  $\Theta(\log N)$**

**Space Complexity:  $\Theta(1)$**

ALGORITHM Adjust(A[],i,N)

BEGIN:

    WHILE 2\*i<=N DO

        j=2\*i

```
    IF j+1<=N THEN
        IF A[j+1]>A[j]
            j=j+1
    IF A[j]>A[i] THEN
        Exchange(A[j],A[i])
    ELSE
        BREAK
```

i=j

**END;**

**Time Complexity:  $\Theta(\log N)$**

**Space Complexity:  $\Theta(1)$**

## CODE

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

struct heap {
    int data[MAX_SIZE];
    int size;
};

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void insert(struct heap *h, int value) {
    if (h->size >= MAX_SIZE) {
        printf("Heap overflow\n");
        return;
    }
    int i = h->size;
    h->data[i] = value;
    while (i > 0 && h->data[i] < h->data[(i-1)/2]) {
        swap(&h->data[i], &h->data[(i-1)/2]);
        i = (i-1)/2;
    }
    h->size++;
}

int delete_minmax(struct heap *h, int is_ascending) {
    if (h->size == 0) {
        printf("Heap underflow\n");
        return -1;
    }
    int root = h->data[0];
    h->data[0] = h->data[h->size-1];
    h->size--;
    int i = 0;
    while (i*2+1 < h->size) {
        int child;
        if (is_ascending) {
            child = (h->data[i*2+1] < h->data[i*2+2] || i*2+2 >= h->size) ? i*2+1 : i*2+2;
            if (h->data[i] > h->data[child]) {
                swap(&h->data[i], &h->data[child]);
                i = child;
            }
        }
        else {

```

```

        break;
    }
}
else {
    child = (h->data[i*2+1] > h->data[i*2+2] || i*2+2 >= h->size) ? i*2+1 : i*2+2;
    if (h->data[i] < h->data[child]) {
        swap(&h->data[i], &h->data[child]);
        i = child;
    }
    else {
        break;
    }
}
}
return root;
}

```

```

void print_heap(struct heap *h) {
    printf("Heap: ");
    for (int i = 0; i < h->size; i++) {
        printf("%d ", h->data[i]);
    }
    printf("\n");
}

```

```

int main() {
    struct heap h;
    h.size = 0;
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");

    insert(&h, 4);
    insert(&h, 10);
    insert(&h, 3);
    insert(&h, 5);
    insert(&h, 1);

    print_heap(&h);

    int min = delete_minmax(&h, 1);
    printf("Deleted minimum element: %d\n", min);

    print_heap(&h);

    int max = delete_minmax(&h, 0);
    printf("Deleted maximum element: %d\n", max);
    print_heap(&h);
}

```



### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Heap: 1 3 4 10 5
Deleted minimum element: 1
Heap: 3 5 4 10
Deleted maximum element: 3
Heap: 10 5 4
```

### Program 84

**Program to Delete kth node from end of a linked list in a single scan and  $O(n)$  time.**

#### Algorithm

Create two pointers, let's call them fast and slow, and initialize both to point to the head of the linked list.

Move the fast pointer k nodes ahead in the linked list.

If the fast pointer becomes NULL before reaching the kth node, it means the linked list is shorter than k nodes, and we cannot delete the kth node from the end. In this case, we return the original linked list.

Move both the fast and slow pointers ahead one node at a time until the fast pointer reaches the end of the linked list.

At this point, the slow pointer points to the (k-1)th node from the end of the linked list. We can delete the kth node by updating the next pointer of the (k-1)th node to skip over the kth node.

Return the head of the modified linked list.

#### CODE

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

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

```
struct node {
    int data;
```

```
    struct node *next;
};
```

```
void deleteKthNodeFromEnd(struct node **head, int k) {
    struct node *p1 = *head;
    struct node *p2 = *head;
    int i;
    for (i = 0; i < k; i++) {
        if (p1 == NULL) {
            printf("The linked list is not long enough.\n");
            return;
        }
        p1 = p1->next;
    }
    while (p1 != NULL) {
        p1 = p1->next;
        p2 = p2->next;
    }
    struct node *temp = p2->next;
    p2->next = p2->next->next;
    free(temp);
}
```

```
void printList(struct node *head) {
    struct node *temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

```
int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    struct node *head = NULL;
    struct node *second = NULL;
    struct node *third = NULL;
    struct node *fourth = NULL;
    struct node *fifth = NULL;

    head = (struct node*) malloc(sizeof(struct node));
    second = (struct node*) malloc(sizeof(struct node));
    third = (struct node*) malloc(sizeof(struct node));
    fourth = (struct node*) malloc(sizeof(struct node));
    fifth = (struct node*) malloc(sizeof(struct node));
}
```

```

head->data = 1;
head->next = second;

second->data = 2;
second->next = third;

third->data = 3;
third->next = fourth;

fourth->data = 4;
fourth->next = fifth;

fifth->data = 5;
fifth->next = NULL;

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

deleteKthNodeFromEnd(&head, 2);
printf("List after deleting the 2nd node from the end: ");
printList(head);

return 0;
}

```

## Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Original list: 1 2 3 4 5
List after deleting the 2nd node from the end: 1 2 3 4
PS C:\Users\91807\Documents\DS Lab>

```

## Experiment 76

**OBJECT**-Program for pair wise swapping of elements of Linked List

**DOMAIN**-Linked List

**ALGORITHM:**

**ALGORITHM** **PairWiseSwap(START)**

**BEGIN:**

```
p=START
WHILE(p!=NULL&& p->next!=NULL) DO
    swap(p->info,p->next->info)
    p=p->next->next
traverse(START)
```

**END;**

TIME COMPLEXITY: $\Theta(N)$

SPACE COMPLEXITY: $\Theta(1)$

### CODE

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

void swapNodes(struct node* prev, struct node* curr) {
    struct node* nextNode = curr->next;
    curr->next = nextNode->next;
    nextNode->next = curr;
    prev->next = nextNode;
}
```

```

void pairWiseSwap(struct node* head) {
    if (head == NULL || head->next == NULL) {
        return;
    }
    struct node* prev = head;
    struct node* curr = head->next;
    struct node* nextNode = curr->next;

    head = curr;
    curr->next = prev;

    while (nextNode != NULL && nextNode->next != NULL) {
        prev->next = nextNode;
        curr->next = nextNode->next;
        nextNode->next = curr;

        prev = curr;
        curr = curr->next;
        nextNode = curr->next;
    }
    if (nextNode != NULL) {
        prev->next = nextNode;
        nextNode->next = curr;
        curr->next = NULL;
    }
}

void printList(struct node* head) {
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

void insertNode(struct node** head, int data) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = data;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
}

```

```

    }
    struct node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

int main() {
    struct node* head = NULL;
    printf("Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024 \n");
    insertNode(&head, 1);
    insertNode(&head, 2);
    insertNode(&head, 3);
    insertNode(&head, 4);
    insertNode(&head, 5);
    printf("Original Linked List: ");
    printLi
    pairWiseSwap(head);
    printf("Linked List after Pairwise Swapping: ");
    printList(head);

    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Original Linked List: 1 2 3 4 5
Linked List after Pairwise Swapping: 1 3 2 5 4

```

## Experiment 75

**OBJECT**-Program for printing the elements of Linked List in Reverse Order

**DOMAIN**-Linked List

**ALGORITHM:**

**ALGORITHM ReverseTraversal(START)**

**BEGIN:**

```
    IF p!=NULL THEN
        ReverseTraversal (p→next)
        WRITE (p→data)
```

**END;**

TIME COMPLEXITY: $\Theta(N)$

SPACE COMPLEXITY: $\Theta(N)$

**CODE**

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

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

```
struct node {
    int data;
    struct node* next;
};
```

```
struct node* newNode(int data) {
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    temp->data = data;
    temp->next = NULL;
    return temp;
}
```

```
void push(struct node** head_ref, int data) {
    struct node* new_node = newNode(data);
```

```

    new_node->next = *head_ref;
    *head_ref = new_node;
}

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

int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");

    struct node* head = NULL;
    push(&head, 1);
    push(&head, 2);
    push(&head, 3);
    push(&head, 4);
    printf("List in reverse order: ");
    printReverse(head);
    printf("\n");
    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
List in reverse order: 1 2 3 4

```



#### Experiment 74

**OBJECT**-Program for reversing contents of Linear Linked List

**DOMAIN**:Linked List

**ALGORITHM**:

**ALGORITHM**

**BEGIN**:

**Reverse Linked LIST(START)**

```
current=START
previous=START->next
WHILE(current!=NULL)
    q=current->next
    current->next=previous
    previous=current
    current=q
START->next=NULL
RETURN START
```

**End;**

TIME COMPLEXITY:  $\Theta(N)$

SPACE COMPLEXITY:  $\Theta(1)$

## CODE

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

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

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

void printList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

void reverseList(struct Node** head) {
    struct Node* previous = NULL;
    struct Node* current = *head;
    struct Node* next;

    while (current != NULL) {
        next = current->next;
        current->next = previous;
        previous = current;
        current = next;
    }
    *head = previous;
}

int main() {
    struct Node* head = NULL;
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
}
```

```
insertAtBeginning(&head, 5);
insertAtBeginning(&head, 4);
insertAtBeginning(&head, 3);
insertAtBeginning(&head, 2);
insertAtBeginning(&head, 1);
printf("Original list: ");
printList(head);
reverseList(&head);
printf("Reversed list: ");
printList(head);
return 0;
}
```

#### Output

Name-Anubhav Singhal \ CS-A \ 2100320120024

Original list: 1 2 3 4 5

Reversed list: 5 4 3 2 1

## Experiment.78

**Object:**program for sorting of Linked List.

**DOMAIN:**Linked List

ALGORITHM: SortingLinked List(START)

Begin: sortingLinked List(START)

```
        p=START
        q=NULL
        temp=0
    WHILE(p->next!=q) DO
        IF(p->info>p->next->info)THEN
            temp=p->info
            p->info=p->next->info
            p->next->info=temp
            p=p->next
            q=p
    END;
```

Space Complexity:  $\Theta(1)$

Time complexity:  $\Theta(n)$

### CODE

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

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

void insert(Node **head, int data) {
    Node *new_node = create_node(data);

    if (*head == NULL) {
        *head = new_node;
```

```

    } else {
        Node *temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = new_node;
    }
}

```

```

void swap(Node *a, Node *b) {
    int temp = a->data;
    a->data = b->data;
    b->data = temp;
}

```

```

void bubble_sort(Node *head) {
    int swapped;
    Node *ptr1;
    Node *lptr = NULL;

    if (head == NULL) {
        return;
    }
    do {
        swapped = 0;
        ptr1 = head;

        while (ptr1->next != lptr) {
            if (ptr1->data > ptr1->next->data) {
                swap(ptr1, ptr1->next);
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    } while (swapped);
}

```

```

void print_list(Node *head) {
    Node *temp = head;

```

```

        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }

int main() {
    Node *head = NULL;
    printf("Name-Anubhav Singhal \\\ CS-A \\\ 2100320120024 \n");
    insert(&head, 5);
    insert(&head, 3);
    insert(&head, 8);
    insert(&head, 1);
    insert(&head, 4);

    printf("Unsorted list: ");
    print_list(head);

    bubble_sort(head);

    printf("Sorted list: ");
    print_list(head);
    return 0;
}

```

### Output

Sorted list: 1 3 4 5 8

### Experiment.79

**Object:** To Detect if there is any cycle in the linked list. (use two pointers, once moves at a speed of one node, other moves at a speed of two nodes. If they collide with each other it means there is a cycle.

**DOMAIN:** Linked List

#### **ALGORITHM:**

ALGORITHM: CycleDetection(START)

Begin:

```
P=START
Q=START
WHILE(1) DO
    IF p→Next==NULL THEN
        RETURN FALSE
    ELSE
        P=p→Next
    IF q→Next==NULL THEN
        RETURN FALSE
    ELSE
        IF q→Next→Next==NULL THEN
            RETURN FALSE
        ELSE
            q=q→Next→Next
    IF p==q THEN
        RETURN TRUE
```

END;

Space Complexity:  $\Theta(N)$

Time complexity:  $\Theta(1)$

## CODE

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

int detectCycle(struct Node* head, struct Node** startNode, int* cycleLength) {
    struct Node* slow = head;
    struct Node* fast = head;
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) {
            *startNode = slow;
            *cycleLength = 1;

            struct Node* ptr1 = slow->next;
            while (ptr1 != slow) {
                ptr1 = ptr1->next;
                (*cycleLength)++;
            }
            return 1;
        }
    }
    return 0;
}

int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 1;

    struct Node* node2 = (struct Node*)malloc(sizeof(struct Node));
    node2->data = 2;
    head->next = node2;
```



```

struct Node* node3 = (struct Node*)malloc(sizeof(struct Node));
node3->data = 3;
node2->next = node3;

struct Node* node4 = (struct Node*)malloc(sizeof(struct Node));
node4->data = 4;
node3->next = node4;

struct Node* node5 = (struct Node*)malloc(sizeof(struct Node));
node5->data = 5;
node4->next = node5;

struct Node* node6 = (struct Node*)malloc(sizeof(struct Node));
node6->data = 6;
node5->next = node6;
node6->next = node3;

struct Node* startNode;
int cycleLength;

if (detectCycle(head, &startNode, &cycleLength)) {
    printf("Cycle detected\n");
    printf("Starting node: %d\n", startNode->data);
    printf("Cycle length: %d\n", cycleLength);
} else {
    printf("No cycle detected\n");
}
return 0;
}

```

## Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Cycle detected
Starting node: 5
Cycle length: 4

```

### Experiment.87

**Object:** Program for polynomial addition using Linked List.

**DOMAIN:** Linked List

#### **ALGORITHM:**

ALGORITHM: PolynomialAdditionLinked List(poly1,poly2)

```
Begin:      poly3=NULL
            p=poly1
            q=poly2
            WHILE(p!=NULL AND q!=NULL) DO
            IF p->Exp==q->Exp THEN
            InsEnd(poly3,p->coeffi+q->coeffi,p->Exp)
            p=p->next
            q=q->next
        ELSE
            IF p->Exp>q->Exp THEN
            InsEnd(poly3,p->coeffi,p->Exp)
            p=p->next
        ELSE
            InsEnd(poly3,q->coeffi,q->Exp)
            q=q->next
        WHILE p!=NULL DO
            InsEnd(poly3,p->coeffi,p->Exp)
            p=p->next
        WHILE q!=NULL DO
            InsEnd(poly3,q->coeffi,p->Exp)
            q=q->next

        RETURN ploy3
END;
```

Space Complexity:  $\Theta(m+n)$

Time complexity:  $\Theta(m+n)$

#### **CODE**

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

```
typedef struct node {
    int coef;
    int exp;
```

```
    struct node *next;
} Node;
```

```
Node* createNode(int coef, int exp) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    newNode->coef = coef;
    newNode->exp = exp;
    newNode->next = NULL;
    return newNode;
}
```

```
void insertNode(Node** head, int coef, int exp) {
    Node* newNode = createNode(coef, exp);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
}
```

```
void display(Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    while (head != NULL) {
        printf("%dx^%d", head->coef, head->exp);
        if (head->next != NULL) {
            printf(" + ");
        }
        head = head->next;
    }
}
```

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

```
Node* add(Node* poly1, Node* poly2) {  
    Node* result = NULL;  
    while (poly1 != NULL && poly2 != NULL) {  
        if (poly1->exp > poly2->exp) {  
            insertNode(&result, poly1->coef, poly1->exp);  
            poly1 = poly1->next;  
        }  
        else if (poly1->exp < poly2->exp) {  
            insertNode(&result, poly2->coef, poly2->exp);  
            poly2 = poly2->next;  
        }  
        else {  
            int coefSum = poly1->coef + poly2->coef;  
            if (coefSum != 0) {  
                insertNode(&result, coefSum, poly1->exp);  
            }  
            poly1 = poly1->next;  
            poly2 = poly2->next;  
        }  
    }  
    while (poly1 != NULL) {  
        insertNode(&result, poly1->coef, poly1->exp);  
        poly1 = poly1->next;  
    }  
    while (poly2 != NULL) {  
        insertNode(&result, poly2->coef, poly2->exp);  
        poly2 = poly2->next;  
    }  
    return result;  
}
```

```
int main() {  
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");  
}
```

```

Node* poly1 = NULL;
insertNode(&poly1, 5, 2);
insertNode(&poly1, 4, 1);
insertNode(&poly1, 2, 0);
printf("Polynomial 1: ");
display(poly1);

Node* poly2 = NULL;
insertNode(&poly2, 3, 2);
insertNode(&poly2, 6, 1);
insertNode(&poly2, 1, 0);
printf("Polynomial 2: ");
display(poly2);

Node* result = add(poly1, poly2);
printf("Resultant polynomial: ");
display(result);

return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Polynomial 1: 5x^2 + 4x^1 + 2x^0
Polynomial 2: 3x^2 + 6x^1 + 1x^0
Resultant polynomial: 8x^2 + 10x^1 + 3x^0

```

## Practical – 82

**PROBLEM STATEMENT :** Program to arrange the consonants and vowel nodes of the linked list in such a way that all the vowel nodes come before the consonants while maintaining the order of their arrival

**ALGORITHM :**

**Begin:**

1. Initialize two pointers "vowel\_ptr" and "consonant\_ptr" to NULL.
2. Traverse the linked list using a pointer "current\_ptr".
3. If the current node's data is a vowel, create a new node "new\_node" with the same data, and append it to the end of the vowel list using the "vowel\_ptr".
4. If the current node's data is a consonant, create a new node "new\_node" with the same data, and append it to the end of the consonant list using the "consonant\_ptr".
5. After traversing the whole list, append the consonant list to the end of the vowel list.
6. Return the new list.

**End;**

## CODE

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

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

```
#include<stdbool.h>
```

```
struct Node
```

```
{
```

```
    char data;
```

```
    struct Node *next;
```

```
};
```

```
struct Node *newNode(char key)
```

```
{
```

```
    struct Node *temp = (struct Node*)malloc(sizeof(struct Node));
```

```
    temp->data = key;
```

```
    temp->next = NULL;
```

```
    return temp;
}
```

```
void printlist(struct Node *head)
{
    if (! head)
    {
        printf("Empty list \n");
        return;
    }
```

```
    while (head != NULL)
    {
        printf("%c",head->data);
        if (head->next)
            printf("->");
        head = head->next;
    }
    printf("\n");
}
```

```
bool isVowel(char x)
{
    return (x == 'a' || x == 'e' || x == 'i' ||
            x == 'o' || x == 'u');
}
```

```

struct Node *arrange(struct Node *head)
{
    struct Node *newHead = head;
    struct Node *latestVowel;
    struct Node *curr = head;

    if (head == NULL)
        return NULL;

    if (isVowel(head->data))
        latestVowel = head;
    else
    {
        while (curr->next != NULL && !isVowel(curr->next->data))
            curr = curr->next;

        if (curr->next == NULL)
            return head;

        latestVowel = newHead = curr->next;
        curr->next = curr->next->next;
        latestVowel->next = head;
    }
    while (curr != NULL && curr->next != NULL)
    {
        if (isVowel(curr->next->data))
        {
            if (curr == latestVowel)
            {

```



```

        latestVowel = curr = curr->next;
    }

    else
    {
        struct Node *temp = latestVowel->next;

        latestVowel->next = curr->next;

        latestVowel = latestVowel->next;

        curr->next = curr->next->next;

        latestVowel->next = temp;
    }
}

else
{
    curr = curr->next;
}
}

return newHead;
}

```

```

int main()
{
    printf("Anubhav Singhal:\n");
    printf("2100320120024:\n");
    printf("CS - A\n");
    struct Node *head = newNode('a');
    head->next = newNode('b');
}

```

```

head->next->next = newNode('c');
head->next->next->next = newNode('e');
head->next->next->next->next = newNode('d');
head->next->next->next->next->next = newNode('o');
head->next->next->next->next->next->next = newNode('x');
head->next->next->next->next->next->next->next = newNode('i');
printf("Linked list before :\n");
printlist(head);
head = arrange(head);
printf("Linked list after :\n");
printlist(head);
return 0;
}

```

## Output

```

- - - - -
Anubhav Singhal:
2100320120024:
CS - A
Linked list before :
a->b->c->e->d->o->x->i
Linked list after :
a->e->o->i->b->c->d->x
-

```

**Practical-80:****Program for Delete duplicate nodes in the Linked List Algorithm:**

- Input the number of elements of the linked list.
- Input the elements of the linked list in sorted order.
- Traverse from the head of the sorted linked list.
- While traversing, compare the current node with the next node.
- If data of the next node is the same as the current node then delete the next node.

**CODE**

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *next;
} *first = NULL;
void Create(int A[], int n)
{
    int i;
    struct Node *t, *last;
    first = (struct Node *) malloc (sizeof (struct Node));
    first->data = A[0];
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        t = (struct Node *) malloc (sizeof (struct Node));
        t->data = A[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
}
```

```

    }
}
void Display(struct Node *p)
{
    while (p != NULL)
    {
        printf ("%d ", p->data);
        p = p->next;
    }
}
void RemoveDuplicate(struct Node *p)
{
    struct Node *q = p->next;
    while (q != NULL)
    {
        if (p->data != q->data)
        {
            p = q;
            q = q->next;
        }
        else
        {
            p->next = q->next;
            free(q);
            q = p->next;
        }
    }
}
int main()
{
    int A[] = { 4, 4, 7, 7, 7 };
    Create (A, 5);
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    printf ("Linked List with Duplicates: \n");
    Display (first);
    RemoveDuplicate (first);
    printf ("\nLinked List without Duplicates: \n");
    Display (first);
    return 0;
}

```

## Output

Name-Anubhav Singhal \ CS-A \ 2100320120024

Linked List with Duplicates:

4 4 7 7 7

Linked List without Duplicates:

4 7

#### **Practical-86: Program to find out the subtraction of two given link list Algorithm:**

- **Step 1:** Check the size of both the linked list.
- **Step 2:** If the size of one linked list is smaller than another one. Then we will append zero at the end of the smaller one to make it of equal length.
- **Step 3:** If the length of both the lists is the same. Then we will calculate which list is smaller in number.
- **Step 4:** After knowing which is the smaller one, we will subtract the nodes of the smaller linked list from the larger linked list one by one. We also have to keep in mind the borrow while doing the difference calculation.

#### **CODE**

```
#include <bits/stdc++.h>
using namespace std;
struct Node { //creation of linked list
    int data;
    struct Node* next;
};
Node* newNode(int data){
    Node* temp = new Node;
    temp->data = data;
    temp->next = NULL;
    return temp;
}
int getLength(Node* Node){ //get the length of both the lists
    int size = 0;
    while (Node != NULL) {
        Node = Node->next;
        size++;
    }
    return size;
}
Node* subtraction(Node* L1, Node* L2, bool& borrow){
```

```

if (L1 == NULL && L2 == NULL && borrow == 0)
return NULL;
Node* previous
= subtraction(
L1 ? L1->next : NULL,
L2 ? L2->next : NULL, borrow);
int d1 = L1->data;
int d2 = L2->data;
int sub = 0;
if (borrow) {
d1--;
borrow = false;
}
if (d1 < d2) {
borrow = true;
d1 = d1 + 10;
}
//subtract the digits
sub = d1 - d2;
Node* current = newNode(sub);
current->next = previous;
return current;
}
//returns the desired result
Node* subtractLinked(Node* L1, Node* L2){
// Base Case.
if (L1 == NULL && L2 == NULL)
return NULL;
int length1 = getLength(L1);
int length2 = getLength(L2);
Node *largerNode = NULL, *smallerNode = NULL;
Node* temp1 = L1;
Node* temp2 = L2;
if (length1 != length2) {
largerNode = length1 > length2 ? L1 : L2;
smallerNode = length1 > length2 ? L2 : L1;
smallerNode = paddZeroes(smallerNode, abs(length1 - length2));
}
else {
while (L1 && L2) {
if (L1->data != L2->data) {
largerNode = L1->data > L2->data ? temp1 : temp2;
smallerNode = L1->data > L2->data ? temp2 : temp1;
break;

```

```

    }
    L1 = L1->next;
    L2 = L2->next;
    }
    }
    if(largerNode==NULL&&smallerNode==NULL){
        return newNode(0);
    }
    bool borrow = false;
    return subtraction(largerNode, smallerNode, borrow);
}
void printList(struct Node* Node)
{
    while (Node != NULL) {
        cout<<Node->data;
        Node = Node->next;
    }
}
int main(){
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \\n");
    Node* head1 = newNode(1);
    head1->next = newNode(0);
    head1->next->next = newNode(0);
    head1->next->next->next = newNode(0);
    Node* head2 = newNode(1);
    head2->next = newNode(0);
    head2->next->next =newNode(0);
    cout<<"AMAN PANDEY ROLLNO 2100320120015\\nCS-A";
    cout<<"\\nLinked List 1: ";
    printList(head1);
    cout<<"\\nLinked List 2: ";
    printList(head2);
    Node* result = subtractLinked(head1, head2);
    cout<<"\\nResultant List is: ";
    printList(result);
    return 0;
}

```

## Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Linked List 1: 1000
Linked List 2: 100
Resultant List is: 0900
```

### Practical-81

#### Linked list implementation of priority queue Algorithm:

1. Define a Node struct to represent each node in the priority queue. The struct should contain the node's value and priority, as well as a pointer to the next node in the list.
2. Define a function insert\_node that takes a pointer to a pointer to the head of the linked list, an integer value, and an integer priority. This function should create a new node with the given value and priority, and insert it into the list in the correct position based on its priority.
3. Define a function remove\_node that takes a pointer to a pointer to the head of the linked list, removes the node with the highest priority from the list, and returns its value.
4. Define a function print\_priority\_queue that takes a pointer to the head of the linked list and prints the contents of the priority queue in order of decreasing priority.
5. To insert a node into the priority queue, call the insert\_node function with the head pointer, value, and priority as arguments
6. To remove a node from the priority queue, call the remove\_node function with the head pointer as an argument.
7. To print the contents of the priority queue, call the print\_priority\_queue function with the head pointer as an argument.

#### CODE

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

```
struct Node
{
```



```
int value;  
int priority;  
struct Node *next;  
};
```

```
void insert_node(struct Node **head, int value, int priority)  
{  
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));  
    new_node->value = value;  
    new_node->priority = priority;  
    if (*head == NULL || priority < (*head)->priority)  
    {  
        new_node->next = *head;  
        *head = new_node;  
    }  
    else  
    {  
        struct Node *current_node = *head;  
        while (current_node->next != NULL && current_node->next->priority <= priority)  
        {  
            current_node = current_node->next;  
        }  
        new_node->next = current_node->next;  
        current_node->next = new_node;  
    }  
}
```

```
int remove_node(struct Node **head)  
{  
    if (*head == NULL)  
    {  
        printf("Error: queue is empty.\n");  
        return -1;  
    }  
    struct Node *node_to_remove = *head;  
    int value = node_to_remove->value;  
    *head = node_to_remove->next;  
    free(node_to_remove);  
    return value;  
}
```

```
void print_priority_queue(struct Node *head)  
{  
    if (head == NULL)  
    {  
        printf("Priority queue is empty.\n");  
        return;  
    }  
}
```

```

}
printf("Priority queue: ");
while (head != NULL)
{
printf("(%d,%d) ", head->value, head->priority);
head = head->next;
}
printf("\n");
}

int main()
{
printf("Name:Anubhav\tRoll NO:2100320120024\nCS-A\n");
struct Node *head = NULL;
insert_node(&head, 1, 2);
insert_node(&head, 2, 1);
insert_node(&head, 3, 3);
print_priority_queue(head);
printf("Removed node with highest priority: %d\n", remove_node(&head));
printf("Removed node with highest priority: %d\n", remove_node(&head));
print_priority_queue(head);
return 0;
}

```

### Output

```

Name:Anubhav      Roll NO:2100320120024
CS-A
Priority queue: (2,1) (1,2) (3,3)
Removed node with highest priority: 2
Removed node with highest priority: 1
Priority queue: (3,3)

```

---

## PRACTICAL – 88

### PROBLEM STATEMENT : POLYNOMIAL MULTIPLICATION BY USING LINKED LIST

#### ALGORITHM :

1. Define a struct for a term in the polynomial, which should include the coefficient and the degree of the term.
2. Create two linked lists to represent the two polynomials to be multiplied.
3. Read in the coefficients and degrees for the two polynomials and create the corresponding linked lists.
4. Define a function to multiply two terms and return a new term.
5. Define a function to insert a term into a linked list, maintaining the order of the terms by degree.
6. Multiply each term in the first polynomial by each term in the second polynomial and insert the resulting terms into a new linked list, combining terms with the same degree.
7. Print out the resulting polynomial.

#### CODE

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Term {
    int coeff;
    int exp;
    struct Term* next;
} Term;

Term* createTerm(int coeff, int exp) {
    Term* newTerm = (Term*)malloc(sizeof(Term));
    newTerm->coeff = coeff;
    newTerm->exp = exp;
    newTerm->next = NULL;
    return newTerm;
}
```

```

void insertTerm(Term** poly, int coeff, int exp) {
    Term* newTerm = createTerm(coeff, exp);
    if (*poly == NULL) {
        *poly = newTerm;
    } else {
        Term* current = *poly;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newTerm;
    }
}

Term* multiplyPolynomials(Term* poly1, Term* poly2) {
    Term* result = NULL;
    Term* current1 = poly1;
    while (current1 != NULL) {
        Term* current2 = poly2;
        while (current2 != NULL) {
            int coeff = current1->coeff * current2->coeff;
            int exp = current1->exp + current2->exp;
            Term* existingTerm = NULL;
            Term* currentResult = result;
            while (currentResult != NULL) {
                if (currentResult->exp == exp) {
                    existingTerm = currentResult;
                    break;
                }
                currentResult = currentResult->next;
            }
            if (existingTerm != NULL) {
                existingTerm->coeff += coeff;
            } else {
                insertTerm(&result, coeff, exp);
            }
            current2 = current2->next;
        }
        current1 = current1->next;
    }
    return result;
}

```

```

void printPolynomial(Term* poly) {
    if (poly == NULL) {
        printf("0\n");
    } else {
        while (poly != NULL) {
            printf("%d*x^%d", poly->coeff, poly->exp);
            poly = poly->next;
        }
    }
}

```

```

        if (poly != NULL) {
            printf(" + ");
        }
    }
    printf("\n");
}
}

int main() {
    printf("Name:Anubhav\tRoll NO:2100320120024\nCS-A\n");
    Term* poly1 = NULL;
    insertTerm(&poly1, 1, 2);
    insertTerm(&poly1, 3, 1);
    insertTerm(&poly1, 2, 0);

    Term* poly2 = NULL;
    insertTerm(&poly2, 2, 1);
    insertTerm(&poly2, 1, 0);

    Term* result = multiplyPolynomials(poly1, poly2);
    printf("Result: ");
    printPolynomial(result);

    free(poly1);
    free(poly2);
    free(result);

    return 0;
}

```

### Output

```

Name:Anubhav    Roll NO:2100320120024
CS-A
Result: 2*x^3 + 7*x^2 + 7*x^1 + 2*x^0

```

## Program 72

**Program for Finding the Middle element of a singly linked list in one pass.**

### Algorithm

1. Initialize two pointers, fast and slow, to the head of the list.
2. Traverse the list with the fast pointer moving two nodes at a time and the slow pointer moving one node at a time, until the fast pointer reaches the end of the list (i.e., fast->next or fast->next->next is NULL).
3. At this point, the slow pointer should be pointing to the middle element of the list (or the first middle element if the list has an even number of elements).
4. Return the value of the middle element or the node itself, depending on the function signature.

### CODE

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

struct Node* getMiddleNode(struct Node* head) {
    struct Node* fast = head;
    struct Node* slow = head;
    while (fast != NULL && fast->next != NULL) {
        fast = fast->next->next;
        slow = slow->next;
    }
    return slow;
}

int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    struct Node* head = NULL;
    struct Node* tail = NULL;
    for (int i = 1; i <= 10; i++) {
        struct Node* node = (struct Node*) malloc(sizeof(struct Node));
        node->data = i;
        node->next = NULL;
        if (head == NULL) {
            head = tail = node;
        } else {
            tail->next = node;
        }
    }
}
```

```

        tail = node;
    }
}
struct Node* middleNode = getMiddleNode(head);
printf("The middle element of the list is %d\n", middleNode->data);
return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
The middle element of the list is 6

```

### Program 73

#### Program to perform Binary Search on the Linked List

#### Algorithm

1. Define a structure for the linked list node, which should contain the value of the node and a pointer to the next node.
2. Create a sorted linked list using the values in ascending order.
3. Define a function for binary search that takes the head of the linked list and the value to search for as input parameters.
4. Initialize two pointers, **low** and **high**, to the head and tail of the linked list, respectively.
5. While **low** is less than or equal to **high**, calculate the midpoint between **low** and **high**.
6. Traverse the linked list from the head node using **mid** iterations.

7. If the value of the node at the **mid** position is equal to the value to search for, return the index of the node.
8. If the value is less than the value to search for, set **low** to **mid+1**.
9. If the value is greater than the value to search for, set **high** to **mid-1**.
10. If the value is not found, return -1.

#### CODE

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

int get_length(struct Node* head) {
    int length = 0;
    while (head != NULL) {
        length++;
        head = head->next;
    }
    return length;
}

int* linked_list_to_array(struct Node* head) {
    int length = get_length(head);
    int* arr = (int*) malloc(length * sizeof(int));
    int i = 0;
    while (head != NULL) {
        arr[i++] = head->data;
        head = head->next;
    }
    return arr;
}

int binary_search(int* arr, int length, int key) {
    int low = 0;
    int high = length - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
}
```



```

    return -1; // key not found
}

int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    struct Node* head = (struct Node*) malloc(sizeof(struct Node));
    head->data = 1;
    head->next = NULL;
    struct Node* prev = head;
    for (int i = 2; i <= 10; i++) {
        struct Node* curr = (struct Node*) malloc(sizeof(struct Node));
        curr->data = i;
        curr->next = NULL;
        prev->next = curr;
        prev = curr;
    }

    int* arr = linked_list_to_array(head);
    int length = get_length(head);

    int key = 5;
    int index = binary_search(arr, length, key);

    if (index == -1) {
        printf("%d not found\n", key);
    } else {
        printf("%d found at index %d\n", key, index);
    }

    free(arr);
    struct Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}

```

## Output

Name-Anubhav Singhal \ CS-A \ 2100320120024  
5 found at index 4

### **Program 83**

#### **Program for Deletion of all occurances of x from Linked List**

##### **Algorithm**

1. Create a temporary pointer current and set it to the head of the linked list.
2. Create a previous pointer and set it to NULL.
3. While current is not NULL, do the following: a. If the data at current is equal to x, set the previous' next pointer to current's next pointer and delete current. b. Else, set the previous pointer to current and move current to the next node.
4. Return the head of the linked list.

##### **CODE**

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

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

```
void deleteX(struct Node** head, int x) {
    struct Node* current = *head;
    struct Node* previous = NULL;
```

```

while (current != NULL) {
    if (current->data == x) {
        if (previous == NULL) {
            *head = current->next;
        } else {
            previous->next = current->next;
        }
        free(current);
        current = previous->next;
    } else {
        previous = current;
        current = current->next;
    }
}
}

void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
}

int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;

    // allocate 3 nodes in the heap
    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 1;
    head->next = second;

    second->data = 2;
    second->next = third;

    third->data = 3;
    third->next = NULL;

    printf("Linked list before deletion: ");
    printList(head);

    int x = 2;
    deleteX(&head, x);

    printf("\nLinked list after deletion of %d: ", x);
    printList(head);

    return 0;
}

```

```
}
```

## OUTPUT

Name-Anubhav Singhal \ CS-A \ 2100320120024

Linked list before deletion: 1 2 3

Linked list after deletion of 2: 1 3

### Program 77

#### Program to find kth node from the last in a single link list

##### Algorithm

1. Start with two pointers, first and second, pointing to the head of the linked list.
2. Move the second pointer k positions ahead.
3. Traverse the linked list with both pointers until the second pointer reaches the end of the list.
4. The first pointer will now be pointing to the kth node from the last.

##### CODE

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

Node* addNode(Node* head, int value) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
    } else {
        Node* last = head;
        while (last->next != NULL) {
            last = last->next;
        }
        last->next = newNode;
    }
}
```

```

    }
    last->next = newNode;
}
return head;
}

```

```

void printList(Node* head) {
    printf("List: ");
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

```

```

Node* findKthNodeFromLast(Node* head, int k) {
    if (head == NULL || k <= 0) {
        return NULL;
    }
    Node* fast = head;
    Node* slow = head;
    int i;
    for (i = 1; i < k && fast != NULL; i++) {
        fast = fast->next;
    }
    if (fast == NULL) {
        return NULL;
    }
    while (fast->next != NULL) {
        fast = fast->next;
        slow = slow->next;
    }
    return slow;
}

```

```

int main() {
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024 \n");
    Node* head = NULL;
    int k, value;
    printf("Enter the values for the list (end with -1):\n");
    do {
        scanf("%d", &value);
        if (value != -1) {
            head = addNode(head, value);
        }
    } while (value != -1);
    printList(head);
    printf("Enter the value of k (positive integer):\n");
    scanf("%d", &k);
    Node* kthNode = findKthNodeFromLast(head, k);
    if (kthNode != NULL) {
        printf("The %d-th node from the last is %d.\n", k, kthNode->data);
    }
}

```

```

    } else {
        printf("There is no %d-th node from the last.\n", k);
    }
    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the values for the list (end with -1):
3 3 2 5 6 -1
List: 3 3 2 5 6
Enter the value of k (positive integer):
5
The 5-th node from the last is 3.

```

### Experiment 89

**OBJECT:**Program for primitive operations of Circular Linked List

**DOMAIN:**Linked List

**ALGORITHM:**

**ALGORITHM InsertBeginning(START, key)**

**BEGIN:**

```

    p=GetNode()
    p→Info=key
    p→Next=START
    START=p

```

**END;**

**Time Complexity:** $\Theta(1)$

The number executed statements are 4 in the above algorithm.

**Space Complexity:** $\Theta(1)$

An extra variable p is used. Also, the space is allocated in the memory to the new node.

**ALGORITHM InsertAfter(p, key)**

**BEGIN:**

```

    q=GetNode()

```

```

q → Info=key
q → Next=p → Next
p → Next=q
END;

```

**Time Complexity:** $\Theta(1)$

The number executed statements are 4 in the above algorithm.

**Space Complexity:** $\Theta(1)$

An extra variable **q** is used. Also, the space is allocated in the memory to the new node.

**ALGORITHM**InsertEnd(**START**, **key**)

**BEGIN:**

```
p=START
```

```

WHILE p!=NULL DO
    p=p → Next

```

```

q = GetNode()
q → Info=key
q → Next=NULL
p → Next=q

```

**END;**

**Time Complexity:** $\Theta(N)$

The while loop runs until the end of the linked list is reached. Hence, for a linked list of **N** items, **N+5** statements will be executed

**Space Complexity:** $\Theta(1)$

Extra variables used here are **p** and **q**. Also, the space is allocated in the memory to the new node

**ALGORITHM** DelBeg(**START**)

**BEGIN:**

```

p=START
START=START → Next
x=p → Info
FreeNode(p)

```

```
    RETURN x
END;
```

**Time Complexity:  $\Theta(1)$**

The above algorithm executes four statements unconditionally

**Space Complexity:  $\Theta(1)$**

Two extra variables, p & x are used

**ALGORITHM DeleteEnd(START)**

**BEGIN:**

```
    p=START
    q=NULL
    WHILE p→Next!=NULL DO
        q=p
        p=p→Next
    q→Next=NULL
    x=p→Info
    FreeNode(p)
    RETURN x
END;
```

**Time Complexity:  $\Theta(N)$**

The while loop executes two statements repetitively until the last element is reached. Also, five more statements are executed outside the loop

**Space Complexity:  $\Theta(1)$**

Three extra variables namely p, q & x are utilized

**ALGORITHM DeleteAft(p)**

**BEGIN:**

```
    IF p == NULL OR p→Next == NULL THEN
        WRITE ("Void Deletion")
        EXIT(1)
    q=p→Next
    x=q→Info
    p→Next=q→Next
```



```
    FreeNode(q)
    RETURN x
END;
```

**Time Complexity:  $\Theta(1)$**

Here, statements are executed

**Space Complexity:  $\Theta(1)$**

Two extra variables q & x are used apart from the parameters passed

**ALGORITHM Traverse(START)**

**BEGIN:**

```
    p=START
    WHILE p!=NULL D
        WRITE p → Info
        p=p → Next
END;
```

**Time Complexity:  $\Theta(N)$**

It takes  $2n$  statement executions by the while loop where  $n$  elements are present in the list. Also, one more statement is executed to initialize  $p$

**Space Complexity:  $\Theta(1)$**

$p$  is taken as an extra variable

**CODE**

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

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

struct node *createNode(int data) {
    struct node *newNode = (struct node*) malloc(sizeof(struct node));
    newNode->data = data;
```

```
    newNode->next = NULL;
    return newNode;
}
```

```
void insertAtBeginning(struct node **head, int data) {
    struct node *newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        (*head)->next = *head;
        return;
    }
    newNode->next = (*head)->next;
    (*head)->next = newNode;
}
```

```
void insertAtEnd(struct node **head, int data) {
    struct node *newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        (*head)->next = *head;
        return;
    }
    struct node *temp = (*head)->next;
    while (temp->next != *head) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->next = *head;
}
```

```
void deleteFromBeginning(struct node **head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
}
```

```

    if ((*head)->next == *head) {
        free(*head);
        *head = NULL;
        return;
    }
    struct node *temp = (*head)->next;
    (*head)->next = temp->next;
    free(temp);
}

void deleteFromEnd(struct node **head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    if ((*head)->next == *head) {
        free(*head);
        *head = NULL;
        return;
    }
    struct node *temp = (*head)->next;
    while (temp->next->next != *head) {
        temp = temp->next;
    }
    struct node *toBeDeleted = temp->next;
    temp->next = *head;
    free(toBeDeleted);
}

void displayList(struct node *head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct node *temp = head->next;

```

```

printf("Circular Linked List: ");
do {
    printf("%d ", temp->data);
    temp = temp->next;
} while (temp != head->next);
printf("\n");
}

int main() {
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024 \n");
    struct node *head = NULL;
    insertAtBeginning(&head, 10);
    insertAtBeginning(&head, 20);
    insertAtEnd(&head, 30);
    displayList(head);
    deleteFromBeginning(&head);
    deleteFromEnd(&head);
    displayList(head);
    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Circular Linked List: 20 30 10
Circular Linked List: 10
PS C:\Users\91807\Documents\DS Lab>

```

## Experiment 90

**OBJECT:**Program for Cocatenation of Circular Linked List

**DOMAIN:**Linked List

**ALGORITHM:**

**ALGORITHM ConcatList(cList1, cList2)**

**BEGIN:**

```
        IF START1==null THEN
            RETURN (START2)
        ELSE
            IF START2==null THEN
                RETURN (START1)
            ELSE
                p=START1->Next
                q=START2->Next
                START1->Next=q
                START2->Next=p
            RETURN (START2)
```

**END;**

Space Complexity:  $\Theta(1)$

Time complexity:  $\Theta(1)$

### CODE

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

void append(struct node **head, int data) {
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = data;
    new_node->next = *head;

    if (*head == NULL) {
        *head = new_node;
        new_node->next = new_node;
        return;
    }

    struct node *last = *head;
    while (last->next != *head)
        last = last->next;
```

```

    last->next = new_node;
    new_node->next = *head;
}

void concat(struct node **head1, struct node **head2) {
    if (*head1 == NULL || *head2 == NULL) {
        printf("Error: Cannot concatenate empty list\n");
        return;
    }
    struct node *last1 = *head1, *last2 = *head2;
    while (last1->next != *head1)
        last1 = last1->next;

    while (last2->next != *head2)
        last2 = last2->next;
    last1->next = *head2;
    last2->next = *head1;
}

void printList(struct node *head) {
    struct node *temp = head;
    if (head != NULL) {
        do {
            printf("%d ", temp->data);
            temp = temp->next;
        } while (temp != head);
    }
    printf("\n");
}

int main() {
    struct node *head1 = NULL, *head2 = NULL;

    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    append(&head1, 1);
    append(&head1, 2);
    append(&head1, 3);
    printf("First circular linked list: ");
    printList(head1);
}

```

```
append(&head2, 4);
append(&head2, 5);
append(&head2, 6);
printf("Second circular linked list: ");
printList(head2);

// Concatenate the two circular linked lists
concat(&head1, &head2);
printf("Concatenated circular linked list: ");
printList(head1);

return 0;
}
```

### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
First circular linked list: 1 2 3
Second circular linked list: 4 5 6
Concatenated circular linked list: 1 2 3 4 5 6
```

## Program 91

### Program for implementation of Josephus Problem.

#### Algorithm

1. Define a structure for each person in the list, containing their ID and a pointer to the next person in the list.
2. Create a function to initialize the list, taking the number of people as input. This function should create a circular linked list of people, with each person's ID set to their position in the list (starting from 1).
3. Create a function to simulate the game, taking the number of people and the "counting out" number as input. This function should iterate through the list, removing every kth person until only one person is left. This can be done by setting the "next" pointer of the person k steps ahead to skip over them, and then deleting the person at the current position.
4. Print the ID of the remaining person as output.

#### CODE

```
#include<stdio.h>
#include<stdlib.h>
#include"cll.h"
void johesph(struct node**CSTART,int k)
{
    struct node*p,*q;
    q=NULL;
    p=*CSTART;
    int cnt=0;
    while(p->next!=p)
    {
        while(cnt!=k)
        {
            q=p;
            cnt++;
            p=p->next;
        }
        p=p->next;
        delafter(&q);
        cnt=1;
    }
    printf("%d",<p->info);
}
int main()
{
```



```

int c=1;
struct node*CSTART;
CSTART=NULL;
CInsBeg(&CSTART,1);
for(int i=2;i<=100;i++)
{
    Cinsend(&CSTART,i);
}
traverse(CSTART);
johesph(&CSTART,13);
printf("\n Name-Anubhav Singhal \ \ CS-A \ \ 2100320120024 \n");
Return 0;
}

```

### Output

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 4
1 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
70
Name-Anubhav Singhal \ CS-A \ 2100320120024

```

## Experiment 92

**OBJECT:**Program for primitive operations of Doubly Linked List

**DOMAIN:**Linked List

**ALGORITHM:**

- **ALGORITHM** InsertBeg(dSTART,key)

**BEGIN:**

```
p=getnode()
info(p)=key
left(p)=NULL
right(p)=dSTART
IF(dSTART!=NULL) THEN
    left(dSTART)=p
    dSTART=p
```

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

- **ALGORITHM Insertend(dSTART,key)**

**BEGIN:**

```
p=dSTART
q=getnode()
info(q)=key
IF(dSTART==NULL) THEN
    insertbeg(dSTART,key)
ELSE
    WHILE(right(p)!=NULL)
        p=right(p)
    right(p)=q
    right(q)=NULL
    left(q)=p
```

**END;**

TIME COMPLEXITY:  $\Theta(N)$

SPACE COMPLEXITY:  $\Theta(1)$

- **ALGORITHM Insertleft(p,key)**

**BEGIN:**

```
IF(left(p)==NULL) THEN
    insertbeg(p,key)
ELSE
    IF(left(p)!=NULL) THEN
        q=getnode()
        r=left(p)
        info(q)=key
        left(q)=r
        right(r)=q
        right(q)=p
        left(p)=q
    ELSE
        insertbeg(p,key)
```

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

- **ALGORITHM Insertright(p,key)**

**BEGIN:**

```

q=getnode()
info(q)=key
r=right(p)
right(p)=q
left(q)=p
IF(right(p)!=NULL) THEN
    right(q)=r
    left(r)=q
ELSE
    right(q)=NULL

```

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

- **ALGORITHM Delbeg(dSTART)**

**BEGIN:**

```

IF(dSTART==NULL) THEN
    write("void deletion")
    exit(1)
ELSE
    p=dSTART
    dSTART=right(dSTART)
    IF(right(p)!=NULL) THEN
        left(dSTART)=NULL
        x=info(p)
        freenode(p)
    RETURN x

```

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

**END;**

- **ALGORITHM DelEnd(dSTART)**

**BEGIN:**

```

IF(dSTART==NULL)
    write("void deletion")
    exit(1)
ELSE
    p=dSTART
    q=NULL
    WHILE(right(p)!=NULL) DO
        q=p
        p=right(p)
        IF(q==NULL) THEN
            dSTART=NULL
        ELSE
            right(q)=NULL

```

```

        x=info(p)
        freenode(p)
        RETURN x
END;
TIME COMPLEXITY:  $\Theta(N)$ 
SPACE COMPLEXITY:  $\Theta(1)$ 

```

- **ALGORITHM DelLeft(p)**

```

BEGIN:
    IF(p==NULL | left(p)==NULL) THEN
        write("void deletion")
        exit(1)
    ELSE
        q=left(p)
        r=left(q)
        right(r)=p
        right(p)=r
        x=info(q)
        freenode(q)
    RETURN x
END;
TIME COMPLEXITY:  $\Theta(1)$ 
SPACE COMPLEXITY:  $\Theta(1)$ 

```

- **ALGORITHM DelRight(p)**

```

BEGIN:
    IF(p==NULL | right(p)==NULL) THEN
        write("void deletion")
        exit(1)
    ELSE
        q=right(p)
        r=right(q)
        right(p)=r
        left(r)=p
        x=info(q)
        freenode(q)
    RETURN x
END;
TIME COMPLEXITY:  $\Theta(1)$ 
SPACE COMPLEXITY:  $\Theta(1)$ 

```

- **ALGORITHM Traverse(dSTART)**

```

BEGIN:
    p=dSTART
    WHILE(p!=NULL) DO
        write(info(p))
        p=right(p)
END;

```

TIME COMPLEXITY:  $\Theta(1)$   
SPACE COMPLEXITY:  $\Theta(1)$

#### CODE

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

// Define the node structure for a doubly linked list
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Initialize an empty list
struct Node* head = NULL;

// Create a new node with the given data
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Insert a new node at the beginning of the list
void insertAtBeginning(int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        head = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}

// Insert a new node at the end of the list
void insertAtEnd(int data) {
    struct Node* newNode = createNode(data);
    struct Node* current = head;
    if (head == NULL) {
        head = newNode;
    } else {
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
        newNode->prev = current;
    }
}
```

```

// Insert a new node after a given node in the list
void insertAfter(struct Node* prevNode, int data) {
    if (prevNode == NULL) {
        printf("Previous node cannot be null.\n");
        return;
    }
    struct Node* newNode = createNode(data);
    newNode->next = prevNode->next;
    prevNode->next = newNode;
    newNode->prev = prevNode;
    if (newNode->next != NULL) {
        newNode->next->prev = newNode;
    }
}

```

```

// Delete the first occurrence of a node with the given data from the list
void deleteNode(int data) {
    struct Node* current = head;
    if (current == NULL) {
        printf("List is empty.\n");
        return;
    }
    if (current->data == data) {
        head = current->next;
        free(current);
        return;
    }
    while (current != NULL && current->data != data) {
        current = current->next;
    }
    if (current == NULL) {
        printf("Node with data %d not found in list.\n", data);
        return;
    }
    current->prev->next = current->next;
    if (current->next != NULL) {
        current->next->prev = current->prev;
    }
    free(current);
}

```

```

// Print the entire list
void printList() {
    struct Node* current = head;
    if (current == NULL) {
        printf("List is empty.\n");
    } else {
        while (current != NULL) {
            printf("%d ", current->data);
            current = current->next;
        }
    }
}

```

```

        printf("\n");
    }
}

// Test the doubly linked list operations
int main() {
printf("Name-Anubhav Singhal \ CS-A \ 2100320120024 \n");
    insertAtBeginning(1);
    insertAtEnd(2);
    insertAtEnd(3);
    insertAfter(head->next, 4);
    deleteNode(2);
    printList();
    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
1 4 3

```

### Practical-85: Program to find addition of two given link.

#### Algorithm

Traverse the two linked lists in order to add preceding zeros in case a list is having lesser digits than the other one. Start from the head node of both lists and call a recursive function for the next nodes. Continue it till the end of the lists. Creates a node for current digits sum and returns the carry.

#### CODE

```

#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
int data;
struct Node* next;
}Node;
Node* newNode(int data)
{
Node* new_node = (Node *)malloc(sizeof(Node));
new_node->data = data;
new_node->next = NULL;
return new_node;
}
void push(Node** head_ref, int new_data)
{
Node* new_node = newNode(new_data);
new_node->next = (*head_ref);

```

```

(*head_ref) = new_node;
}
Node* addTwoLists(Node* first, Node* second){
Node* res = NULL;
Node *temp, *prev = NULL;
int carry = 0, sum;
while (first != NULL || second != NULL) {
sum = carry + (first ? first->data : 0) + (second ? second->data : 0);
carry = (sum >= 10) ? 1 : 0;
sum = sum % 10;
temp = newNode(sum);
if (res == NULL)
res = temp;
else
prev->next = temp;
prev = temp;
if (first)
first = first->next;
if (second)
second = second->next;
}
if (carry > 0)
temp->next = newNode(carry);
return res;
}
Node* reverse(Node* head){
if (head == NULL || head->next == NULL)
return head;
Node* rest = reverse(head->next);
head->next->next = head;
head->next = NULL;
return rest;
}
void printList(Node* node){
while (node != NULL) {
printf("%d ",node->data);
node = node->next;
}
printf("\n");
}
int main(void){
Node* res = NULL;
Node* first = NULL;
Node* second = NULL;
printf("Anubhav Roll.No: 2100320120024\n CS-A");
push(&first, 6);
push(&first, 4);
push(&first, 9);
push(&first, 5);
push(&first, 7);
printf("First list is ");
printList(first);

```



```

push(&second, 4);
push(&second, 8);
printf("Second list is ");
printList(second);
first = reverse(first);
second = reverse(second);
res = addTwoLists(first, second);
res = reverse(res);
printf("Resultant list is ");
printList(res);
return 0;
}

```

### Output

```

Anubhav Roll.No: 2100320120024
CS-AFirst list is 7 5 9 4 6
Second list is 8 4
Resultant list is 7 6 0 3 0

```

### Experiment 93

**OBJECT-Program for primitive operations of Circular Doubly Linked List**  
**DOMAIN:Linked List**

#### ALGORITHM:

- **ALGORITHM InsertEnd**

#### BEGIN:

```

p=d.START
q=getnode()
IF(d.START==NULL) THEN
    insertbeg()
ELSE
    right(q)=right(p)
    left(q)=p
    left(p)=q
    info(q)=key
    d.START=q

```

#### END;

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

- **ALGORITHM DeleteBeginning**

#### BEGIN:

```

IF(d.START=NULL) THEN
    write("void deletion")
ELSE

```

```

    p=d.START
    q=right(p)
    left(p)=right(q)
    IF(right(q)!=q) THEN
        left(right(q))=p
        x=info(p)
    ELSE
        d.START=NULL
        x=info(p)
    RETURN x
    free q

```

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

- **ALGORITHM InsertLeft(p,key)**

**BEGIN:**

```

    IF(p==NULL) THEN

    ELSE
        IF(left(p)==NULL) THEN
            insertbeg()
        ELSE
            q=getnode()
            left(q)=left(p)
            right(q)=p
            info(q)=key
            right(left(p))=q

```

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

- **ALGORITHM InsertRight(p,key)**

**BEGIN:**

```

    IF(p==NULL)
        insertbeg()
    ELSE
        q=getnode()
        left(q)=p
        right(q)=right(p)
        left(right(p))=q
        right(p)=q

```

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

- **ALGORITHM Traverse(cdSTART)**

**BEGIN:**

```

        p=cdSTART
        WHILE(p!=NULL) DO
            write(info(p))
            p=right(p)
    END;
    TIME COMPLEXITY:  $\Theta(1)$ 
    SPACE COMPLEXITY:  $\Theta(1)$ 

```

### CODE

```

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

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

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

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        (*head)->prev = *head;
        (*head)->next = *head;
        return;
    }
    newNode->prev = (*head)->prev;
    newNode->next = *head;
    (*head)->prev->next = newNode;
    (*head)->prev = newNode;
    *head = newNode;
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        (*head)->prev = *head;
        (*head)->next = *head;
    }
}

```

```

        return;
    }
    newNode->prev = (*head)->prev;
    newNode->next = *head;
    (*head)->prev->next = newNode;
    (*head)->prev = newNode;

void deleteAtBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = *head;
    (*head)->prev->next = (*head)->next;
    (*head)->next->prev = (*head)->prev;
    *head = (*head)->next;
    free(temp);
}

void deleteAtEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = (*head)->prev;
    (*head)->prev = temp->prev;
    temp->prev->next = *head;
    free(temp);
}

void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    do {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data;
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");

```

```

    printf("1. Insert at beginning\n2. Insert at end\n3. Delete at beginning\n4. Delete at end\n5.
Display\n6. Exit\n ");
    while (1) {
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter data: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;
            case 2:
                printf("Enter data: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;
            case 3:
                deleteAtBeginning(&head);
                break;
            case 4:
                deleteAtEnd(&head);
                break;
            case 5:
                display(head);
                break;
            case 6:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }
    return 0;
}

```

## Output

Name-Anubhav Singhal \ CS-A \ 2100320120024

1. Insert at beginning
2. Insert at end
3. Delete at beginning
4. Delete at end
5. Display
6. Exit

Enter your choice:1

Enter data: 3

Enter your choice:1

Enter data: 4

Enter your choice:2

Enter data: 5

Enter your choice:2

Enter data: 6

Enter your choice:3

Enter your choice:4

Enter your choice:5

3 5

Enter your choice:6

#### Experiment 95

**OBJECT:**Program for linear linked list implementation of Linear Queue

**DOMAIN:**Linked List

**ALGORITHM:**

**ALGORITHM Initialize(FRONT,REAR)**

**BEGIN:**

**REAR=NULL**

**FRONT=NULL**

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

**ALGORITHM Empty(FRONT)**

**BEGIN:**

**IF(FRONT==NULL) THEN**

**RETURN TRUE**

**ELSE**

**RETURN FALSE**

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

**ALGORITHM ENQUEUE(FRONT,REAR,x)**

**BEGIN:**

**IF(REAR==NULL) THEN**

        insertbeg(REAR,x)

**FRONT=REAR**

```

        ELSE
            insertsafter(REAR,x)
            REAR=REAR(next)
    END;
    TIME COMPLEXITY:  $\Theta(1)$ 
    SPACE COMPLEXITY:  $\Theta(1)$ 

```

#### **ALGORITHM Dequeue(FRONT,REAR)**

```

    BEGIN:
        IF(FRONT==NULL) THEN
            write("void deletion")
            exit(1)
        ELSE
            x=delbeg(FRONT)
            IF(FRONT==NULL)
                REAR=NULL
        RETURN x
    END;
    TIME COMPLEXITY:  $\Theta(1)$ 
    SPACE COMPLEXITY:  $\Theta(1)$ 

```

#### **ALGORITHM Traverse(FRONT)**

```

    BEGIN:
        WHILE FRONT!=NULL
            write(FRONT(info) )
            FRONT=FRONT(next)
    END;
    TIME COMPLEXITY:  $\Theta(1)$ 
    SPACE COMPLEXITY:  $\Theta(1)$ 

```

#### **CODE**

```

#include <stdio.h>
#include <stdlib.h>
typedef struct queue_node {
    int data;
    struct queue_node *next;
} QueueNode;

typedef struct {
    QueueNode *front;
    QueueNode *rear;
} Queue;

void initializeQueue(Queue *q) {
    q->front = NULL;
    q->rear = NULL;
}

```

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

```
void enqueue(Queue *q, int data) {
    // Create a new node
    QueueNode *newNode = (QueueNode *) malloc(sizeof(QueueNode));
    if (newNode == NULL) {
        printf("Memory allocation error\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;

    if (isEmpty(q)) {
        q->front = newNode;
        q->rear = newNode;
    }
    else {
        q->rear->next = newNode;
        q->rear = newNode;
    }
}
```

```
int dequeue(Queue *q) {
    int data;
    QueueNode *temp;

    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    else {
        temp = q->front;
        data = temp->data;
        q->front = q->front->next;
        free(temp);
        return data;
    }
}
```

```
void display(Queue *q) {
    QueueNode *temp = q->front;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}
```



```
    printf("\n");  
}  
  
int main() {  
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");  
    Queue q;  
    initializeQueue(&q);  
    enqueue(&q, 10);  
    enqueue(&q, 20);  
    enqueue(&q, 30);  
  
    printf("Queue: ");  
    display(&q);  
  
    dequeue(&q);  
    printf("Queue after dequeue: ");  
    display(&q);  
  
    return 0;  
}
```

#### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
Queue: 10 20 30  
Queue after dequeue: 20 30  
PS C:\Users\91807\Documents\DS Lab>
```

## Experiment 96

**OBJECT:**Program for linear linked list implementation of Double Ended Queue

**DOMAIN:**Linked List

**ALGORITHM:**

**ALGORITHM ENQUEUE(x)**

**BEGIN:**

```
    q=getnode()
    IF (REAR==NULL) THEN
        q(next)=q
    ELSE
        q(next)=REAR(next)
        REAR(next)=q
    REAR=q
```

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

**ALGORITHM Dequeue()**

**BEGIN:**

```
    IF (REAR==NULL) THEN
        write("void deletion")
        exit(1)
    x=REAR.next.data
    IF REAR.next==REAR)
        REAR=NULL
    ELSE
        REAR.next=REAR.next.next
    RETURN x
```

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

## CODE

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

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

```
typedef struct DequeNode {
    int data;
    struct DequeNode* next;
    struct DequeNode* prev;
} DequeNode;
```

```
typedef struct Deque {
    DequeNode* front;
    DequeNode* rear;
} Deque;
```

```
DequeNode* createDequeNode(int data) {
    DequeNode* newNode = (DequeNode*)malloc(sizeof(DequeNode));
```

```

    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

void initDeque(Deque* deque) {
    deque->front = NULL;
    deque->rear = NULL;
}

int isDequeEmpty(Deque* deque) {
    return deque->front == NULL;
}

void addFront(Deque* deque, int data) {
    DequeNode* newNode = createDequeNode(data);
    if (isDequeEmpty(deque)) {
        deque->front = newNode;
        deque->rear = newNode;
    } else {
        newNode->next = deque->front;
        deque->front->prev = newNode;
        deque->front = newNode;
    }
}

void addRear(Deque* deque, int data) {
    DequeNode* newNode = createDequeNode(data);
    if (isDequeEmpty(deque)) {
        deque->front = newNode;
        deque->rear = newNode;
    } else {
        newNode->prev = deque->rear;
        deque->rear->next = newNode;
        deque->rear = newNode;
    }
}

int removeFront(Deque* deque) {
    if (isDequeEmpty(deque)) {
        printf("Deque is empty.\n");
        return -1;
    } else {
        int data = deque->front->data;
        DequeNode* temp = deque->front;
        deque->front = deque->front->next;
    }
}

```

```

        if (deque->front != NULL) {
            deque->front->prev = NULL;
        } else {
            deque->rear = NULL;
        }
        free(temp);
        return data;
    }
}

```

```

int removeRear(Deque* deque) {
    if (isEmpty(deque)) {
        printf("Deque is empty.\n");
        return -1;
    } else {
        int data = deque->rear->data;
        DequeNode* temp = deque->rear;
        deque->rear = deque->rear->prev;
        if (deque->rear != NULL) {
            deque->rear->next = NULL;
        } else {
            deque->front = NULL;
        }
        free(temp);
        return data;
    }
}

```

```

void printDeque(Deque* deque) {
    if (isEmpty(deque)) {
        printf("Deque is empty.\n");
    } else {
        DequeNode* temp = deque->front;
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}

```

```

int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    Deque deque;
    initDeque(&deque);
    addFront(&deque, 10);
    addFront(&deque, 20);
    addRear(&deque, 30);
}

```

```
    addFront(&deque, 40);  
    removeFront(&deque);  
    removeRear(&deque);  
    printDeque(&deque);  
    return 0;  
}
```

### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
20 10
```

### Experiment 94

**OBJECT:**Program for linked list implementation of stack

**DOMAIN:**Linked List

**ALGORITHM** Initialize(FRONT,REAR)

**BEGIN:**

    top=NULL

**END;**

TIME COMPLEXITY:  $\Theta(1)$

SPACE COMPLEXITY:  $\Theta(1)$

**ALGORITHM** Push(top,x)

**BEGIN:**

```
        insertbeg(top,x)
END;
```

#### **ALGORITHM Pop(top)**

```
BEGIN:
    IF(top==NULL) THEN
        write("underflow")
        exit(1)
    ELSE
        x=delbeg(top)
    RETURN x
END;
```

#### **ALGORITHM Empty(top)**

```
BEGIN:
    IF(top==NULL) THEN
        RETURN TRUE
    ELSE
        RETURN FALSE
END;
```

#### **ALGORITHM StackTop(top)**

```
BEGIN:
    RETURN(info(top))
END;
```

#### **CODE**

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

struct Node* top = NULL;

void push(int x) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = x;
    temp->next = top;
    top = temp;
}

void pop() {
    if (top == NULL) {
        printf("Stack is empty\n");
        return;
    }
    struct Node* temp = top;
    top = top->next;
```

```

    free(temp);
}

void display() {
    struct Node* temp = top;
    printf("Stack: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024 \n");
    push(5);
    push(10);
    push(15);
    display();
    pop();
    display();
    push(20);
    display();
    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Stack: 15 10 5
Stack: 10 5
Stack: 20 10 5

```

### Experiment 97

**OBJECT:**Program for recursive creation and traversal of Binary Tree and traversal

**DOMAIN:**Tree

**ALGORITHM:**

**ALGORITHM PreorderTraversal(root)**

**BEGIN:**

```

    IF root != NULL THEN
        WRITE(Data(root))
        PreorderTraversal(Left(root))
        PreorderTraversal(Right(root))

```

**END;**

TIME COMPLEXITY:  $\Theta(N)$

SPACE COMPLEXITY:  $\Theta(\log N)$  for balanced trees

**ALGORITHM PostorderTraversal(root)**

**BEGIN:**

```
    IF root != NULL THEN
        PostorderTraversal(Left(root))
        PostorderTraversal(Right(root))
        WRITE(Data(root))
```

**END;**

TIME COMPLEXITY:  $\Theta(N)$

SPACE COMPLEXITY:  $\Theta(\log N)$  for balanced trees

**ALGORITHM InorderTraversal(root)**

**BEGIN:**

```
    WHILE root != NULL THEN
        InorderTraversal(Left(root))
        WRITE(Data(root))
        InorderTraversal(Right(root))
```

**END;**

TIME COMPLEXITY:  $\Theta(N)$

SPACE COMPLEXITY:  $\Theta(\log N)$  for balanced trees

**ALGORITHM CreateTree(tree)**

**BEGIN:**

```
    WRITE("Whether Left of DATA(tree) exists (1/0)")
    READ(choice)
    IF (choice == 1) THEN
        WRITE("Input the information of Left node")
        READ(x)
        p ← MakeNode(x)
        LEFT(tree) ← p
        CreateTree(p)
```

```
    WRITE("Whether Right of DATA(tree) exists (1/0)")
    READ(choice)
```

```
    IF (choice == 1) THEN
        WRITE("Input the information of Right node")
        READ(x)
        p ← MakeNode(x)
        Right(tree) ← p
        CreateTree(p)
```

**END;**



**Time Complexity:  $\Theta(N)$** 

There are N elements for creating nodes.

**Space Complexity:  $\Theta(N)$** 

There are N nodes created

**CODE**

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

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

void preOrderTraversal(struct node* node) {
    if (node != NULL) {
        printf("%d ", node->data);
        preOrderTraversal(node->left);
        preOrderTraversal(node->right);
    }
}

void inOrderTraversal(struct node* node) {
    if (node != NULL) {
        inOrderTraversal(node->left);
        printf("%d ", node->data);
        inOrderTraversal(node->right);
    }
}
```

```

    }
}

void postOrderTraversal(struct node* node) {
    if (node != NULL) {
        postOrderTraversal(node->left);
        postOrderTraversal(node->right);
        printf("%d ", node->data);
    }
}

int main() {
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024 \n");
    struct node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    printf("Pre-order traversal: ");
    preOrderTraversal(root);
    printf("\n");
    printf("In-order traversal: ");
    inOrderTraversal(root);
    printf("\n");
    printf("Post-order traversal: ");
    postOrderTraversal(root);
    printf("\n");
    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Pre-order traversal: 1 2 4 5 3
In-order traversal: 4 2 5 1 3
Post-order traversal: 4 5 2 3 1

```

Experiment 98,101

**Object:** Program for creation of Binary Tree recursively and finding its height  
**DOMAIN:**Tree

## ALGORITHM:

### ALGORITHM HEIGHT(root)

#### BEGIN:

```
    IF root==NULL THEN
        RETURN 0
    ELSE
        IF Left(root)==NULL AND Right(root)==NULL
            RETURN 0
        ELSE
            RETURN 1+ Max(HEIGHT(Left(root)),Height(Right(root)))
```

#### END;

TIME COMPLEXITY:  $\Theta$  (Log N) for balanced Trees. N for Skewed Trees

SPACE COMPLEXITY:  $\Theta$  (1)

#### CODE

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

struct node *create_node(int data) {
    struct node *new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}

struct node *create_binary_tree() {
    int data;
    struct node *root = NULL;
    printf("Enter data (or -1 for no node): ");
    scanf("%d", &data);
    if (data == -1) {
        return NULL;
    }
    root = create_node(data);
    printf("Enter left child of %d:\n", data);
    root->left = create_binary_tree();
    printf("Enter right child of %d:\n", data);
    root->right = create_binary_tree();
    return root;
}

int find_height(struct node *root) {
    if (root == NULL) {
```

```

        return -1;
    }
    int left_height = find_height(root->left);
    int right_height = find_height(root->right);
    return 1 + (left_height > right_height ? left_height : right_height);
}

int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \\n");
    struct node *root = NULL;
    root = create_binary_tree();
    printf("Height of binary tree: %d\\n", find_height(root));
    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter data (or -1 for no node): 1
Enter left child of 1:
Enter data (or -1 for no node): 2
Enter left child of 2:
Enter data (or -1 for no node): 3
Enter left child of 3:
Enter data (or -1 for no node): -1
Enter right child of 3:
Enter data (or -1 for no node): 4
Enter left child of 4:
Enter data (or -1 for no node): -1
Enter right child of 4:
Enter data (or -1 for no node): -1
Enter right child of 2:
Enter data (or -1 for no node): 5
Enter left child of 5:
Enter data (or -1 for no node): -1
Enter right child of 5:
Enter data (or -1 for no node): -1
Enter right child of 1:
Enter data (or -1 for no node): 6
Enter left child of 6:
Enter data (or -1 for no node): -1
Enter right child of 6:
Enter data (or -1 for no node): -1
Height of binary tree: 3

```

### Experiment 100

**Object:** Program to find count of nodes having 1 child, 2 children and leaf nodes

**DOMAIN:**Tree

**ALGORITHM:**

**ALGORITHM CountOfN1(root)**

**BEGIN:**

```
    IF tree == NULL THEN
        RETURN 0
    ELSE
        IF LEFT(tree) == NULL && Right(tree) == NULL THEN
            RETURN 0
        ELSE
            IF LEFT(tree) != NULL && Right(tree) != NULL THEN
                RETURN CountOfN1(LEFT(tree))+CountOfN1(Right(tree));
            ELSE
                RETURN 1+CountOfN1(LEFT(tree))+CountOfN1(Right(tree))
```

**ALGORITHM CountOfN2(tree)**

**BEGIN:**

```
    IF tree == NULL THEN
        RETURN 0
    ELSE
        IF LEFT(tree) == NULL && Right(tree) == NULL THEN
            RETURN 0
        ELSE
            IF LEFT(tree) != NULL && Right(tree) != NULL THEN
                RETURN 1+ CountOfN1(LEFT(tree))+CountOfN1(Right(tree))
            ELSE
                RETURN CountOfN1(LEFT(tree))+CountOfN1(Right(tree))
```

**ALGORITHM CountOfN0(root)**

**BEGIN:**

```
    IF tree == NULL THEN
        RETURN 0
    ELSE
```

```

IF LEFT(tree) == NULL && Right(tree) == NULL THEN
    RETURN 1
ELSE
    RETURN CountOfN0(LEFT(tree))+CountOfN0(Right(tree));

```

### Time Complexity: $\Theta(N)$

Need to reach all the node once for computing height

### Space Complexity: $\Omega(\log_2 N)$ , $O(N)$

If the tree is balanced, Call stack will have  $\log_2 n + 1$  entries to the maximum at any moment in case the tree is balanced. If the tree is skewed then the call stack can be grow up to  $n$  activation records

### CODE

```

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

```

```

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

```

```

void countNodes(struct node *root, int *oneChild, int *twoChild, int *leaf) {
    if (root == NULL) {
        return;
    }
}

```

```

    if (root->left != NULL && root->right != NULL) {
        (*twoChild)++;
    } else if (root->left != NULL || root->right != NULL) {
        (*oneChild)++;
    } else {
        (*leaf)++;
    }
    countNodes(root->left, oneChild, twoChild, leaf);
    countNodes(root->right, oneChild, twoChild, leaf);
}

int main() {
    printf("Name-Anubhav Singhal \\\ CS-A \\\ 2100320120024 \n");
    struct node *root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->left = createNode(6);

    int oneChild = 0, twoChild = 0, leaf = 0;
    countNodes(root, &oneChild, &twoChild, &leaf);
    printf("One child nodes: %d\n", oneChild);
    printf("Two child nodes: %d\n", twoChild);
    printf("Leaf nodes: %d\n", leaf);

    return 0;
}

```

## Output

Name-Anubhav Singhal \ CS-A \ 2100320120024

One child nodes: 1

Two child nodes: 2

Leaf nodes: 3



### Experiment 103

**Object: Program for Finding if the Binary Tree is Complete**

**DOMAIN:Tree**

**ALGORITHM:**

**ALGORITHM isComplete (root, index, number\_nodes)**

**BEGIN:**

```
    IF tree == NULL THEN
        RETURN True;
        // If index assigned to current node is more than
        // number of nodes in tree, then tree is not complete
    IF index >= number_nodes THEN
        RETURN False;
        // Recursive for Left and Right subtrees
    RETURN (isComplete(LEFT(root), 2*index + 1, number_nodes) AND isComplete(Right(root),
    2*index + 2, number_nodes))
```

**END;**

**Time Complexity:  $\Theta(N)$**

Need to reach all the node once for computing height

**Space Complexity:  $\Omega(\log_2 N)$ ,  $O(N)$**

If the tree is balanced, Call stack will have  $\log_2 n + 1$  entries to the maximum at any moment in case the tree is balanced. If the tree is skewed then the call stack can be grow upto n activation records

#### **Output**

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
struct Node{
int key;
struct Node *left, *right;
};
struct Node *newNode(char k)
{
struct Node *node = (struct Node*)malloc(sizeof(struct Node));
node->key = k;
node->right = node->left = NULL;
return node;
}
```

```

unsigned int countNodes(struct Node* root)
{
    if (root == NULL)
        return (0);
    return (1 + countNodes(root->left) + countNodes(root->right));
}

bool isComplete (struct Node* root, unsigned int index,
unsigned int number_nodes){
    // An empty tree is complete
    if (root == NULL)
        return (true);
    if (index >= number_nodes)
        return (false);
    return (isComplete(root->left, 2*index + 1, number_nodes) &&
isComplete(root->right, 2*index + 2, number_nodes));
}

int main(){
    struct Node* root = NULL;
    root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    printf("Anubhav Roll.No: 2100320120024\n CS-A\n");
    unsigned int node_count = countNodes(root);
    unsigned int index = 0;
    if (isComplete(root, index, node_count))
        printf("The Binary Tree is complete\n");
    else
        printf("The Binary Tree is not complete\n");
    return (0);
}

```

### Output

```

Anubhav Roll.No: 2100320120024
CS-A
The Binary Tree is complete

```

### Node count in the binary tree Algorithm.

The countNodes() function uses recursion to count the number of nodes in the binary tree. If the root node is NULL, it returns 0. Otherwise, it returns 1 (for the root node) plus the count of nodes in the left and right subtrees. The base case is when the function reaches a leaf node (where both left and right children are NULL), which contributes 1 to the count. The algorithm has a time complexity of  $O(n)$  where  $n$  is the number of nodes in the binary tree.

### CODE

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

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

int countNodes(struct node *root)
{
    {
        if (root == NULL)
        {
            return 0;
        }
        else
        {
            return 1 + countNodes(root->left) + countNodes(root->right);
        }
    }
}

int main()
{
    printf("Name:Anubhav\tRoll NO:2100320120024\nCS-A\n");
    struct node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    printf("The number of nodes in the binary tree is %d.\n", countNodes(root));
    return 0;
}
```

**Output**

Name: Anubhav      Roll NO: 2100320120024

CS-A

The number of nodes in the binary tree is 5.

**Practical-104****Program to find if the given Binary Tree is strictly Algorithm•**

Construct a Binary Tree. • Traverse through the tree in Depth-First-Search (DFS) manner

• Check if the node has 0 children. If yes, return true.

- Check if the node has 2 children. If yes, check the validity of both the children and return the answer.
- If neither of the cases is true, the node contains 1 child. Return false in this case since the tree is not a Strict Binary Tree.

### CODE

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

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

int isStrictlyBinary(struct node* root) {
    if (root == NULL)
        return 1;
    else if (root->left == NULL && root->right == NULL)
        return 1;
    else if (root->left != NULL && root->right != NULL)
        return (isStrictlyBinary(root->left) && isStrictlyBinary(root->right));
    else
        return 0;
}

int main() {
    printf("Anubhav Roll.No: 2100320120024\n CS-A\n");
    struct node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    if (isStrictlyBinary(root))
        printf("The binary tree is strictly binary.\n");
    else
        printf("The binary tree is not strictly binary.\n");
    return 0;
}
```

**Output**

Anubhav Roll.No: 2100320120024

CS-A

The binary tree is strictly binary.

**Practical-110**

**Program to find the diameter of the Binary Tree (distance between the farthest node)**

**Algorithm:**

- 1) Node passed in function is null then return zero.
- 2) Find the height of the left subtree.
- 3) Find the height of the right subtree.
- 4) Using recursive call, calculate the diameter of left-subtree until node becomes NULL.

- 5) Using recursive call, calculate the diameter of right-subtree until node becomes NULL.
- 6) If the root node is included then, Diameter = left-subtree + right-subtree + 1.
- 7) If the root node is not included then, Diameter = max(diameter of left-subtree, diameter of right subtree) .
- 8) The final output will return the max of step 6 and step 7

## CODE CODE

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

```
struct node {
    int data;
    struct node *left;
    struct node *right;
};
```

```
struct node* getNewNode(int data) {
    struct node* newNode =(struct node*)malloc(sizeof(struct node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
```

```
struct node* generateBTree(){
    struct node* root = getNewNode(1);
    root->left = getNewNode(2);
    root->right = getNewNode(3);
    root->left->left = getNewNode(4);
    root->left->right = getNewNode(5);
    root->right->left = getNewNode(6);
    root->right->right = getNewNode(7);
    root->left->left->left = getNewNode(8);
    return root;
}
```

```
int getMax(int a, int b){
    if(a >= b)
```

```

return a;
else
return b;
}

int getHeight(struct node *root){
    int leftHeight, rightHeight;
    if(root == NULL)
        return 0;
    leftHeight = getHeight(root->left);
    rightHeight = getHeight(root->right);
    return getMax(leftHeight, rightHeight) + 1;
}

int getDiameter(struct node *nodePtr) {
    if (nodePtr == NULL)
        return 0;
    int leftHeight = getHeight(nodePtr->left);
    int rightHeight = getHeight(nodePtr->right);
    int leftDiameter = getDiameter(nodePtr->left);
    int rightDiameter = getDiameter(nodePtr->right);
    return getMax(leftHeight + rightHeight + 1,
        getMax(leftDiameter, rightDiameter));
}

int main()
{
    printf("Anubhav Roll.No: 2100320120024\n CS-A\n");
    struct node *root = generateBTree();
    printf("Diameter of Tree = %d", getDiameter(root));
    getchar();
    return 0;
}

```

## Output



Anubhav Roll.No: 2100320120024  
CS-A  
Diameter of Tree = 6

#### Experiment 105

**Object:** Program for Level Order Traversal of Binary Tree

**DOMAIN:**Tree

**ALGORITHM:**

**ALGORITHM LevelOrderTraversal(root)**

**BEGIN:**

```
    Queue Q
    Initialize(Q)
    Enqueue(root)
    WHILE !Empty(Q) DO
        x ← DeQueue(Q)
        WRITE(Data(x))
        IF LEFT(x) !=NULL THEN
            EnQueue(x)
        IF Right(x) !=NULL THEN
            EnQueue(x)
```

**END;**

**Time Complexity:  $\Theta(N)$**

Each node is inserted in the queue once and deleted from queue once. Total of  $2*N$  operations of constant time.

**Space Complexity:  $\Theta(N)$**

Queue size is  $N$

#### CODE

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

```

struct Node* right;
};

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

void levelOrderTraversal(struct Node* root) {
if (root == NULL) {
return;
}
struct Node* queue[100];
int front = 0;
int rear = 0;
queue[rear] = root;
rear++;
while (front < rear) {
struct Node* current = queue[front];
front++;
printf("%d ", current->data);
if (current->left != NULL) {
queue[rear] = current->left;
rear++;
}
if (current->right != NULL) {
queue[rear] = current->right;
rear++;
}
}
}

int main() {
printf("Name-Anubhav Singhal \\\ CS-A \\\ 2100320120024 \n");
struct Node* root = newNode(1);
root->left = newNode(2);
root->right = newNode(3);
root->left->left = newNode(4);
root->left->right = newNode(5);
root->right->left = newNode(6);
root->right->right = newNode(7);
printf("Level Order Traversal: ");

```

```

levelOrderTraversal(root);
return 0;
}

```

**Output**

LEVEL ORDER TRAVERSAL: 1 2 3 4 5 6 7

## Experiment 109

**Object:** Program for BST Insertion , search operations in BST.

**DOMAIN:** BST

**ALGORITHM:**

**ALGORITHM BSTInsert(Tree,key)**

**BEGIN:**

```

    P = Tree
    Q = Null
    R = MakeNode(key)
    WHILE P!=NULL Do
        IF key < Data(P)
            Q = P
            P = Left(P)
        ELSE
            Q = P
            P = Right(P)
        IF key < Data(Q) THEN
            Left(Q) = R
            Father(R) = Q
        ELSE
            Right(Q) = R
            Father(R) = Q
    
```

**END;**

**Time Complexity:**  $\Theta(\log N)$  if Tree is balanced

**Space Complexity:**  $\Theta(1)$

**CODE**

```

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

```

```

typedef struct node{
    int key;
    struct node *left, *right;
}Node;

```

```

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

```

```

Node *insert(Node *root, int key){
    if(root == NULL){
        return createNode(key);
    }
    Node *cur = root, *prev = NULL;
    while(cur != NULL){
        prev = cur;
        if(key < cur->key){
            cur = cur->left;
        }else if(key > cur->key){
            cur = cur->right;
        }else{
            return root;
        }
    }
    if(key < prev->key){
        prev->left = createNode(key);
    }else{
        prev->right = createNode(key);
    }
    return root;
}

```

```

Node *search(Node *root, int key){
    Node *cur = root;
    while(cur != NULL){
        if(key < cur->key){
            cur = cur->left;
        }else if(key > cur->key){
            cur = cur->right;
        }else{
            return cur;
        }
    }
    return NULL;
}

```

```

void inOrder(Node *root){
    if(root != NULL){
        inOrder(root->left);
        printf("%d ", root->key);
    }
}

```

```

        inOrder(root->right);
    }
}

int main(){
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024 \n");
    Node *root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    printf("Inorder traversal of BST: ");
    inOrder(root);
    printf("\n");

    int key = 40;
    Node *searchResult = search(root, key);
    if(searchResult != NULL){
        printf("%d is found in the BST\n", key);
    }else{
        printf("%d is not found in the BST\n", key);
    }

    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Inorder traversal of BST: 20 30 40 50 60 70 80
40 is found in the BST

```

## Experiment 111

**Object:**Program for BST Deletion

**DOMAIN:** BST

**ALGORITHM:**

**ALGORITHM BSTDelete(p)**

**BEGIN:**

```
    IF Left(p)==NULL AND RIGHT(q)==NULL THEN
        IF IsLeft(p) THEN
            Left(Father(p))=NULL
        ELSE
            Right(Father(p))=NULL
        X=Data(p)
        FreeNode(p)
        RETURN X
    ELSE
        IF Left(p) == NULL THEN
            q=Right(p)
            IF IsLeft(p) THEN
                Left(Father(p))=q
                Father(q)=Father(p)
            ELSE
                Right(Father(p))=q
                Father(q)=Father(p)
        ELSE
            IF Right(p) == NULL THEN
                q=Right(p)
                IF IsLeft(p) THEN
                    Left(Father(p))=q
                    Father(q)=Father(p)
                ELSE
                    Right(Father(p))=q
                    Father(q)=Father(p)
                X=Data(p)
                FreeNode(p)
                RETURN X
            ELSE
                q=BSTSuccessor(p)
                y=BSTDelete(q)
                x=Data(p)
                Data(p)=y
                RETURN
            Right(Q) = R
            Father(R) = Q
```

**END;**

**Time Complexity:**  $\Theta(\log N)$  if Tree is balanced

**Space Complexity:**  $\Theta(1)$

## CODE

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

struct node* create_node(int value) {
    struct node* new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = value;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}

struct node* insert_node(struct node* root, int value) {
    if(root == NULL) {
        return create_node(value);
    }
    if(value < root->data) {
        root->left = insert_node(root->left, value);
    }
    else {
        root->right = insert_node(root->right, value);
    }
    return root;
}

struct node* find_minimum_node(struct node* node) {
    struct node* current_node = node;
    while(current_node && current_node->left != NULL) {
        current_node = current_node->left;
    }
    return current_node;
}

struct node* delete_node(struct node* root, int value) {
    if(root == NULL) {
        return root;
    }
    if(value < root->data) {
        root->left = delete_node(root->left, value);
    }
    else if(value > root->data) {
        root->right = delete_node(root->right, value);
    }
    else {
```

```

    if(root->left == NULL) {
        struct node* temp_node = root->right;
        free(root);
        return temp_node;
    }
    else if(root->right == NULL) {
        struct node* temp_node = root->left;
        free(root);
        return temp_node;
    }
    struct node* temp_node = find_minimum_node(root->right);
    root->data = temp_node->data;
    root->right = delete_node(root->right, temp_node->data);
}
return root;
}

```

```

void inorder_traversal(struct node* root) {
    if(root != NULL) {
        inorder_traversal(root->left);
        printf("%d ", root->data);
        inorder_traversal(root->right);
    }
}

```

```

int main() {
    printf("Name-Anubhav Singhal || CS-A || 2100320120024 \n");
    struct node* root = NULL;
    root = insert_node(root, 50);
    root = insert_node(root, 30);
    root = insert_node(root, 20);
    root = insert_node(root, 40);
    root = insert_node(root, 70);
    root = insert_node(root, 60);
    root = insert_node(root, 80);

    printf("BST before deletion: ");
    inorder_traversal(root);

    root = delete_node(root, 20);
    printf("\nBST after deletion of 20: ");
    inorder_traversal(root);

    root = delete_node(root, 30);
    printf("\nBST after deletion of 30: ");
    inorder_traversal(root);

    root = delete_node(root, 50);
    printf("\nBST after deletion of 50: ");
    inorder_traversal(root);
}

```



```
    return 0;  
}
```

### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
BST before deletion: 20 30 40 50 60 70 80  
BST after deletion of 20: 30 40 50 60 70 80  
BST after deletion of 30: 40 50 60 70 80  
BST after deletion of 50: 40 60 70 80
```

### Experiment 113

**Object:**Program for insertion in AVL Tree

**DOMAIN:**AVL Tree

**ALGORITHM:**

**ALGORITHM AVLInsertion(N, Key)**

**BEGIN:**

```
    IF N == NULL THEN  
        N = MakeNode(Key)  
    ELSE  
        IF Key < N→Data THEN  
            N→Left = AVLInsertion(N→Left, Key)  
            IF Balance(N) == 2 THEN  
                IF Key < N→Left→Data  
                    N = LL(N)  
                ELSE  
                    N = LR(N)  
            ELSE  
                N→Right = AVLInsertion(N→Right, Key)  
                IF Balance(N) == -2 THEN  
                    IF Key > N→Left→Data  
                        N = RR(N)  
                    ELSE  
                        N = RL(N)  
                ELSE  
                    N→h = Height(N)  
                RETURN N  
            END;
```

**Time Complexity:**  $\Omega(\log 2N)$ ,  $O(N)$

**Space Complexity:**  $\Theta(1)$

**CODE**

```
#include <stdio.h>
#include <stdlib.h>
typedef struct AVLNode {
    int value;
    int height;
    struct AVLNode* left;
    struct AVLNode* right;
} AVLNode;

int max(int a, int b) {
    return (a > b) ? a : b;
}

int getHeight(AVLNode* node) {
    if (node == NULL) {
        return -1;
    }
    return node->height;
}

int getBalanceFactor(AVLNode* node) {
    if (node == NULL) {
        return 0;
    }
    return getHeight(node->left) - getHeight(node->right);
}

AVLNode* rotateRight(AVLNode* y) {
    AVLNode* x = y->left;
    AVLNode* t2 = x->right;
    x->right = y;
    y->left = t2;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;
```

```

    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;
    return x;
}

```

```

AVLNode* rotateLeft(AVLNode* x) {
    AVLNode* y = x->right;
    AVLNode* t2 = y->left;
    y->left = x;
    x->right = t2;
    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;

    return y;
}

```

```

AVLNode* insert(AVLNode* node, int value) {
    if (node == NULL) {
        AVLNode* new_node = (AVLNode*) malloc(sizeof(AVLNode));
        new_node->value = value;
        new_node->height = 0;
        new_node->left = NULL;
        new_node->right = NULL;
        return new_node;
    }
    if (value < node->value) {
        node->left = insert(node->left, value);
    } else if (value > node->value) {
        node->right = insert(node->right, value);
    } else {
        return node;
    }
    node->height = 1 + max(getHeight(node->left), getHeight(node->right));
    int balance_factor = getBalanceFactor(node);
    if (balance_factor > 1 && value < node->left->value) {
        return rotateRight(node);
    }
}

```

```

    }
    if (balance_factor < -1 && value > node->right->value) {
        return rotateLeft(node);
    }
    if (balance_factor > 1 && value > node->left->value) {
        node->left = rotateLeft(node->left);
        return rotateRight(node);
    }
    if (balance_factor < -1 && value < node->right->value) {
        node->right = rotateRight(node->right);
        return rotateLeft(node);
    }

    return node;
}

```

```

void printInOrder(AVLNode* node) {
    if (node == NULL) {
        return;
    }
    printInOrder(node->left);
    printf("%d ", node->value);
    printInOrder(node->right);
}

```

```

int main() {
    AVLNode* root = NULL;
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024 \n");
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
}

```

```
    root = insert(root, 25);

    printf("In-order traversal: ");
    printInOrder(root);
    return 0;
}
```

### Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
In-order traversal: 10 20 25 30 40 50
-----
```

## Experiment 116

**Object:**Program for implementation of BFS

**DOMAIN:**Graph

**ALGORITHM:**

**ALGORITHM BFS (G, s)**

**BEGIN:**

    QUEUE Q

    Initialize (Q)

    For all  $u \in V[G]$  DO

$\Pi[u] \leftarrow \text{Nil}$

$\text{Color}[u] \leftarrow \text{White}$

    EnQueue (Q, s)

$\text{Color}[s] \leftarrow \text{Grey}$

$d[s] \leftarrow 0$

    WHILE (!Empty(Q)) Do

$u \leftarrow \text{DeQueue (Q)}$  Do

        For each  $V \in \text{Adj [u]}$

            If  $\text{Color[ u]} == \text{White}$

$\text{Color [ u]} \leftarrow \text{Grey}$

                EnQueue (Q, u)

$d[v] \leftarrow d[u]+1$

$\Pi[u] \leftarrow u$

$\text{Color}[u] \leftarrow \text{Black}$

        WRITE (u)

**END;**

**Time Complexity:**  $\Theta(|V|+|E|)$

**Space Complexity:**  $\Theta(|V|)$

### CODE

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

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

```
#define MAX_VERTICES 100
```

```
struct queue {
```

```
    int items[MAX_VERTICES];
```

```
    int front;
```

```

    int rear;
};

struct queue* createQueue() {
    struct queue* q = (struct queue*)malloc(sizeof(struct queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

int isEmpty(struct queue* q) {
    if (q->rear == -1) {
        return 1;
    } else {
        return 0;
    }
}

void enqueue(struct queue* q, int value) {
    if (q->rear == MAX_VERTICES - 1) {
        printf("Queue is full\n");
    } else {
        if (q->front == -1) {
            q->front = 0;
        }
        q->rear++;
        q->items[q->rear] = value;
    }
}

int dequeue(struct queue* q) {
    int item;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        item = -1;
    } else {
        item = q->items[q->front];
        q->front++;
        if (q->front > q->rear) {
            q->front = q->rear = -1;
        }
    }
    return item;
}

void createGraph(int adjacencyMatrix[][MAX_VERTICES], int n) {
    int i, j;
    for (i = 0; i < n; i++) {

```

```

        for (j = 0; j < n; j++) {
            printf("Enter the edge weight between node %d and %d: ", i, j);
            scanf("%d", &adjacencyMatrix[i][j]);
        }
    }
}

```

```

void breadthFirstSearch(int adjacencyMatrix[][MAX_VERTICES], int n, int startingNode) {
    int visited[MAX_VERTICES] = {0};
    struct queue* q = createQueue();
    visited[startingNode] = 1;
    enqueue(q, startingNode);
    printf("Breadth First Search starting from node %d: ", startingNode);
    while (!isEmpty(q)) {
        int currentNode = dequeue(q);
        printf("%d ", currentNode);
        int i;
        for (i = 0; i < n; i++) {
            if (adjacencyMatrix[currentNode][i] && !visited[i]) {
                visited[i] = 1;
                enqueue(q, i);
            }
        }
    }
}

```

```

int main() {
    int n, startingNode;
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \n");
    int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the number of nodes in the graph: ");
    scanf("%d", &n);
    createGraph(adjacencyMatrix, n);
    printf("Enter the starting node: ");
    scanf("%d", &startingNode);
    breadthFirstSearch(adjacencyMatrix, n, startingNode);
    printf("\n");
    return 0;
}

```



**Output**

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the number of nodes in the graph: 4
Enter the edge weight between node 0 and 0: 2
Enter the edge weight between node 0 and 1: 1
Enter the edge weight between node 0 and 2: 3
Enter the edge weight between node 0 and 3: 2
Enter the edge weight between node 1 and 0: 4
Enter the edge weight between node 1 and 1: 3
Enter the edge weight between node 1 and 2: 22
Enter the edge weight between node 1 and 3: 2
Enter the edge weight between node 2 and 0: 5
Enter the edge weight between node 2 and 1: 7
Enter the edge weight between node 2 and 2: 6
Enter the edge weight between node 2 and 3: 5
Enter the edge weight between node 3 and 0: 4
Enter the edge weight between node 3 and 1: 1
Enter the edge weight between node 3 and 2: 9
Enter the edge weight between node 3 and 3: 6
Enter the starting node: 1
Breadth First Search starting from node 1: 1 0 2 3
```

**Object:**Program for DFS on graph

**DOMAIN:**Graph

**ALGORITHM:**

**ALGORITHM DFS (G)**

**BEGIN:**

For all  $u \in V[G]$  Do

Color[u]  $\leftarrow$  White

Time  $\leftarrow$  0

For all  $u \in V[u]$  Do

IF Color[u]  $\leftarrow$  White

DFS  $\leftarrow$  Visit (u)

**END;**

**ALGORITHM DFS-VISIT (U)**

**BEGIN:**

Color[u]  $\leftarrow$  Grey

S[u]  $\leftarrow$  Time+1

For all  $V \in \text{Adj}[U]$  Do

IF Color [V]  $\leftarrow$  White

$\Pi[V] \leftarrow U$

DFS  $\leftarrow$  Visit [V]

Color [U]  $\leftarrow$  Black

F[U]  $\leftarrow$  Time+1

Time  $\leftarrow$  Time+1

**END;**

**Time Complexity:**  $\Theta(|V|+|E|)$

**Space Complexity:**  $\Theta(|V|)$

**CODE**

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

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

```
#define MAX 100
```

```

int visited[MAX];
int adj[MAX][MAX];
int n;

void dfs(int v) {
    visited[v] = 1; // mark vertex as visited
    printf("%d ", v); // print vertex
    for (int i = 0; i < n; i++) {
        if (adj[v][i] == 1 && visited[i] == 0) {
            dfs(i);
        }
    }
}

int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \\n");
    int start; // starting vertex for DFS
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix:\\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }
    printf("Enter starting vertex: ");
    scanf("%d", &start);

    printf("DFS traversal: ");
    dfs(start);
    return 0;
}

```

## Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter number of vertices: 3
Enter adjacency matrix:
1 2 3 4 5 6 7 8 9
Enter starting vertex: 2
DFS traversal: 2
```

---

## Experiment 119

**Object:**Program for All pairs Shortest Path using Floyd-Warshall ALGORITHM

**DOMAIN:**Graph

**ALGORITHM:**

**ALGORITHM APSPFloydWarshall ( W [ ] [ ], N)**

**BEGIN:**

```
    FOR i ← 1 To N Do
        FOR j ← 1 To N DO
            IF W [i] [ j] = 0
                IF i != j THEN
                    W [i] [j]= ∞
        FOR k ← 1 To N Do
            FOR i ← 1 To N DO
                FOR j ← 1 To N DO
                    W [i] [j] ← Min (W [i] [j] , W [i] [k]+ W [k] [j])
```

**END ;**

**Time Complexity:**  $\Theta(N^3)$

**Space Complexity:**  $\Theta(1)$

## CODE

```
#include <stdio.h>
#define MAX_VERTICES 100

int graph[MAX_VERTICES][MAX_VERTICES];
int dist[MAX_VERTICES][MAX_VERTICES];
```

```

void warshall(int vertices) {
    int i, j, k;
    for (k = 0; k < vertices; k++) {
        for (i = 0; i < vertices; i++) {
            for (j = 0; j < vertices; j++) {
                if (graph[i][j] || (graph[i][k] && graph[k][j])) {
                    graph[i][j] = 1;
                }
            }
        }
    }
}

```

```

for (i = 0; i < vertices; i++) {
    for (j = 0; j < vertices; j++) {
        if (graph[i][j]) {
            dist[i][j] = 1;
        }
    }
}
}

```

```

int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \\n");
    int vertices, edges, i, j, u, v;
    printf("Enter the number of vertices and edges: ");
    scanf("%d%d", &vertices, &edges);

    for (i = 0; i < vertices; i++) {
        for (j = 0; j < vertices; j++) {
            graph[i][j] = 0;
        }
    }

    for (i = 0; i < edges; i++) {

```

```

        printf("Enter edge %d (u v): ", i + 1);
        scanf("%d%d", &u, &v);
        graph[u][v] = 1;
    }

    warshall(vertices);

    printf("\nAll Pairs Shortest Path:\n");
    for (i = 0; i < vertices; i++) {
        for (j = 0; j < vertices; j++) {
            printf("%d ", dist[i][j]);
        }
        printf("\n");
    }
    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Enter the number of vertices and edges: 2 2
Enter edge 1 (u v): 3 6
Enter edge 2 (u v): 5 2

All Pairs Shortest Path:
0 0
0 0

```

### **PRACTICAL 107**

**Program to build expression tree from the given infix expression.**

#### **.ALGORITHM**

If T is not NULL:

    If T->data is an operand:

        return T.data

A = solve(T.left)

B = solve(T.right)

--> Calculate operator for 'T.data' on A and B, and call recursively,

    return calculate(A, B, T.data)

#### **CODE**

```
#include <stdio.h>
#include <stdlib.h>
struct node {char data;
struct node* left;
struct node* right;
struct node* next;};
struct node *head=NULL;
struct node* newNode(char data)
{struct node* node
= (struct node*)malloc(sizeof(struct node));
node->data = data;
node->left = NULL;
node->right = NULL;
```

```

node->next = NULL;
return (node);}
void printInorder(struct node* node)
{if (node == NULL)
return;
else{printInorder(node->left);
printf("%c ", node->data);
printInorder(node->right);}}
void push(struct node* x)
{if(head==NULL)
head = x;
else
{(x)->next = head;
head = x;}}
struct node* pop()
{struct node* p = head;
head = head->next;
return p;}
int main()
{printf("Anubhav Roll.No: 2100320120024\n CS-A\n");
char s[] = {'A','B','C','*','+','D','/'};
int l = sizeof(s) / sizeof(s[0]) ;
struct node *x, *y, *z;
for (int i = 0; i < l; i++) {
if (s[i] == '+' || s[i] == '-' || s[i] == '*'
|| s[i] == '/' || s[i] == '^') {
z = newNode(s[i]);
x = pop();
y = pop();
z->left = y;
z->right = x;
push(z);}
else { z = newNode(s[i]);
push(z);}}
printf(" The Inorder Traversal of Expression Tree: ");
printInorder(z);
return 0;}

```

### Output

Anubhav Roll.No: 2100320120024

CS-A

The Inorder Traversal of Expression Tree: A + B \* C / D



### **Program 118**

**Program to find the number of connected components in the undirected Graph.**

#### **Algorithm**

1. Initialize a counter variable to 0 for counting the number of connected components.
2. For each node in the graph, do the following steps: a. If the node is not visited yet, perform a depth-first search (DFS) from the node to find all the nodes that are reachable from it. b. Increment the counter variable by 1 since all the reachable nodes are part of a new connected component.
3. Return the counter variable as the total number of connected components in the graph.

#### **Time Complexity**

**$O(V + E)$** , where V is the number of vertices and E is the number of edges in the graph.

#### **CODE**

```
#include <stdio.h>
#define MAX_SIZE 100

int adj[MAX_SIZE][MAX_SIZE];
int visited[MAX_SIZE];
int n;

void dfs(int v) {
    visited[v] = 1;
```

```

    for (int i = 0; i < n; i++) {
        if (adj[v][i] && !visited[i]) {
            dfs(i);
        }
    }
}

```

```

int count_connected_components() {
    int count = 0;
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i);
            count++;
        }
    }
    return count;
}

```

```

int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \\n");
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }
    int count = count_connected_components();
    printf("Number of connected components: %d\\n", count);
    return 0;
}

```

## Output

Name-Anubhav Singhal \ CS-A \ 2100320120024

Enter the number of vertices: 3

Enter the adjacency matrix:

1 2 3 4 5 6 7 8 9

Number of connected components: 1

### Program 106

**Write a program to create a copy of the given Binary Tree**

#### Algorithm

1. Create a new node with the same value as the root of the given tree.
2. Recursively copy the left subtree of the given tree to the left subtree of the new node.
3. Recursively copy the right subtree of the given tree to the right subtree of the new node.
4. Return the new tree.

#### CODE

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

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

```
struct node {  
    int data;  
    struct node *left;  
    struct node *right;  
};
```

```
struct node* create_node(int data) {  
    struct node* new_node = (struct node*)malloc(sizeof(struct node));  
    new_node->data = data;  
    new_node->left = NULL;  
    new_node->right = NULL;
```

```
return new_node;
}
```

```
struct node* copy_tree(struct node* root) {
    if (root == NULL) {
        return NULL;
    }
    struct node* new_node = create_node(root->data);
    new_node->left = copy_tree(root->left);
    new_node->right = copy_tree(root->right);
    return new_node;
}
```

```
void print_tree(struct node* root) {
    if (root == NULL) {
        return;
    }
    printf("%d ", root->data);
    print_tree(root->left);
    print_tree(root->right);
}
```

```
int main() {
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \\n");
    struct node* root = create_node(1);
    root->left = create_node(2);
    root->right = create_node(3);
    root->left->left = create_node(4);
    root->left->right = create_node(5);
    root->right->left = create_node(6);
    root->right->right = create_node(7);
    struct node* new_tree = copy_tree(root);
```

```
    printf("Original tree: ");
    print_tree(root);
    printf("\\nCopied tree: ");
    print_tree(new_tree);
```

```
    return 0;
}
```

## Output

Name-Anubhav Singhal \ CS-A \ 2100320120024

Original tree: 1 2 4 5 3 6 7

Copied tree: 1 2 4 5 3 6 7

## Program 108

write a program to check if the given tree is BST or not.

### Algorithm

1. Create a function isBST(node) that returns true if the given node is a valid BST
2. If the node is null, return true
3. Check if the left subtree is a BST and if the maximum value in the left subtree is less than the node's value
4. Check if the right subtree is a BST and if the minimum value in the right subtree is greater than the node's value
5. If both conditions are true, return true, else return false.

### CODE

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

```
struct node {
    int data;
    struct node* left;
    struct node* right;
};
```

```
struct node* newNode(int data) {
    struct node* node = (struct node*)malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
```

```
}
```

```
bool isBSTUtil(struct node* node, int min, int max) {  
    if (node == NULL)  
        return true;
```

```
    if (node->data < min || node->data > max)  
        return false;
```

```
    return isBSTUtil(node->left, min, node->data - 1) && isBSTUtil(node->right, node->data + 1,  
max);  
}
```

```
bool isBST(struct node* root) {  
    return isBSTUtil(root, INT_MIN, INT_MAX);  
}
```

```
int main() {  
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \\n");  
    struct node* root = newNode(4);  
    root->left = newNode(2);  
    root->right = newNode(5);  
    root->left->left = newNode(1);  
    root->left->right = newNode(3);
```

```
    if (isBST(root))  
        printf("The given tree is a BST\\n");  
    else  
        printf("The given tree is not a BST\\n");
```

```
    return 0;  
}
```

## Output

```
Name-Anubhav Singhal \ CS-A \ 2100320120024  
The given tree is a BST
```

### **Program 102**

**write a program or function to find the sum all nodes in a given binary tree.**

#### **Algorithm**

1. Define a function to calculate the sum of all nodes in a binary tree.
2. Check if the root is None, return 0.
3. Calculate the sum of left subtree recursively.
4. Calculate the sum of right subtree recursively.
5. Add the root value to the sum of left subtree and right subtree.
6. Return the total sum.

#### **CODE**

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

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

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

```
int sumTreeNodes(struct TreeNode* root) {  
    if (root == NULL) {  
        return 0;  
    }  
    else {  
        return (root->val + sumTreeNodes(root->left) + sumTreeNodes(root->right));  
    }  
}
```

```
int main() {  
    printf("Name-Anubhav Singhal \\ CS-A \\ 2100320120024 \\n");  
    struct TreeNode* root = (struct TreeNode*) malloc(sizeof(struct TreeNode));  
    root->val = 1;  
    root->left = (struct TreeNode*) malloc(sizeof(struct TreeNode));
```

Sum of all nodes in the binary tree: 15



## Program 112

### Write a Program for BST insertion (using Recursion).

#### Algorithm

1. Create a function to insert a new node in the BST.
2. Check if the root is NULL, if it is, create a new node and set it as the root.
3. If the new value is less than the current node's value, recursively call the function on the left subtree.
4. If the new value is greater than the current node's value, recursively call the function on the right subtree.
5. Repeat steps 3-4 until a leaf node is reached.
6. Create a new node with the new value and set it as the child of the leaf node.

#### CODE

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

```
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
```

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

```
struct Node* insertNode(struct Node* root, int value) {
    if (root == NULL) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insertNode(root->left, value);
    }
}
```

```

    else if (value > root->data) {
        root->right = insertNode(root->right, value);
    }
    return root;
}

void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

int main() {
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024 \n");
    struct Node* root = NULL;
    root = insertNode(root, 50);
    insertNode(root, 30);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
    insertNode(root, 60);
    insertNode(root, 80);

    printf("Inorder Traversal of BST: ");
    inorderTraversal(root);

    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
Inorder Traversal of BST: 20 30 40 50 60 70 80

```

## Program120

### Program For Linked List Implementation of General Sparse Matrix

#### Algorithm

1. Define the structure of a node in the linked list, which should contain three elements: row, column and value.
2. Define a function to create a new node in the linked list, which takes three arguments: row, column and value. The function should allocate memory for the node and initialize its elements with the given values.
3. Define a function to insert a new node into the linked list. The function should take the head of the linked list, row, column and value as arguments. It should create a new node with the given values and insert it into the appropriate position in the linked list.
4. Define a function to print the linked list in the matrix form, which takes the head of the linked list and the size of the matrix as arguments. The function should iterate over the rows and columns of the matrix and print the value of the node at the corresponding position in the linked list. If there is no node at that position, it should print zero.
5. Define a function to add two sparse matrices using linked list representation. The function should take the heads of the two linked lists and the size of the matrix as arguments. It should create a new linked list to store the result of the addition and return its head.

#### CODE

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

struct Node {
    int row;
    int col;
    int val;
    struct Node* next;
};

struct SparseMatrix {
    int rows;
    int cols;
    struct Node** rowsArr;
};

struct Node* createNode(int row, int col, int val) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->row = row;
    newNode->col = col;
```

```

    newNode->val = val;
    newNode->next = NULL;
    return newNode;
}

struct SparseMatrix* createSparseMatrix(int rows, int cols) {
    struct SparseMatrix* matrix = (struct SparseMatrix*)malloc(sizeof(struct SparseMatrix));
    matrix->rows = rows;
    matrix->cols = cols;
    matrix->rowsArr = (struct Node**)calloc(rows, sizeof(struct Node*));
    return matrix;
}

void insertElement(struct SparseMatrix* matrix, int row, int col, int val) {
    if (row >= matrix->rows || col >= matrix->cols) {
        printf("Error: Index out of range.\n");
        return;
    }

    struct Node* newNode = createNode(row, col, val);
    struct Node* current = matrix->rowsArr[row];

    if (current == NULL) {
        matrix->rowsArr[row] = newNode;
    }
    else {
        while (current->next != NULL && current->next->col < col) {
            current = current->next;
        }
        newNode->next = current->next;
        current->next = newNode;
    }
}

void printSparseMatrix(struct SparseMatrix* matrix) {
    for (int i = 0; i < matrix->rows; i++) {
        struct Node* current = matrix->rowsArr[i];
        for (int j = 0; j < matrix->cols; j++) {
            if (current != NULL && current->col == j) {
                printf("%d ", current->val);
                current = current->next;
            }
            else {
                printf("0 ");
            }
        }
    }
}

```

```

    }
    printf("\n");
}

int main() {
    printf("Name-Anubhav Singhal \ CS-A \ 2100320120024 \n");
    struct SparseMatrix* matrix = createSparseMatrix(3, 3);

    insertElement(matrix, 0, 0, 1);
    insertElement(matrix, 0, 2, 2);
    insertElement(matrix, 1, 1, 3);
    insertElement(matrix, 2, 0, 4);
    insertElement(matrix, 2, 2, 5);

    printSparseMatrix(matrix);

    return 0;
}

```

### Output

```

Name-Anubhav Singhal \ CS-A \ 2100320120024
1 0 2
0 3 0
4 0 5

```