

PRACTICAL FILE



Data and File Structures (MCA-162)

Submitted by:

Hitesh Walia
Roll no.: 00311104422
Sem: MCA 2nd
Batch: 2022-24

Submitted to:

Dr. Anu Taneja
Associate Professor

INDEX

S. No	Question	T. Sign
1.	Write a program in C to implement insertion in an array.	
2.	Write a program in C to implement deletion in an array.	
3.	Write a program in C to find largest and 2 nd largest element in array.	
4.	Write a program in C to implement Linear Search.	
5.	Write a program in C to implement Linear search with multiple duplicate keys.	
6.	Write a program in C to implement binary search.	
7.	Write a program in C to implement binary search with recursion.	
8.	Write a program in C to implement bubble sort.	
9.	Write a program in C to implement insertion sort.	
10.	Write a program in C to implement selection sort.	
11.	Write a program in C to implement Counting sort.	
12.	Write a program in C to implement Radix sort.	
13.	Write a menu driven program to add two matrices, to subtract two matrices and to multiply two matrices.	
14.	Write a program to check two matrices are identical or not.	
15.	Write a program to check whether given matrix is identity or not.	
16.	Write a program to check given matrix is sparse or not.	
17.	Write a program to convert sparse matrix in row triplet form.	

18.	Write a program to print all diagonal value of matrix.	
19.	Write a program to sort all rows in ascending order.	
20.	Write a program to print sum of all rows in matrix.	
21.	Write a program to find missing element in an array.	
22.	Write a program to remove duplicates from an array.	
23.	Write a program to print duplicate elements in an array.	
24.	Write a program to create a dynamic array and find sum of elements of an array using malloc(), calloc(), realloc() and free() functions.	
25.	Write a program to create array of structures of 5 employees with their EmpId, Ename and Salary.Display all details of the employee with highest salary.	
26.	Write a program to perform following operations on singly linked list:- Create, Traverse, Insertion and Deletion(beg, end , specific), Length of list, Searching, Reverse list.	
27.	Write a program to perform following operations on doubly linked list:- Create, Traverse, Insertion and Deletion(beg, end , specific), Length of list, Reverse list.	
28.	Write a program to perform following operations on singly circular linked list:- Create, Traverse, Insertion and Deletion(beg, end , specific), Length of list, Reverse list.	
29.	Write a program to perform following operations on doubly circular linked list:- Create, Traverse, Insertion and Deletion(beg, end , specific), Length of list, Reverse list.	
30.	Write a program to print middle element of the linked list.	
31.	Write a program to check if a linked list is palindrome or not.	
32.	Write a program to add and multiply two polynomials using linked list.	
33.	Write a program to sort a linked list and add a node in that sorted linked list.	

34.	Write a program to implement Stack as an Array.	
35.	Write a program to implement Stack as a Linked List.	
36.	Write a program to Reverse a String using Stack.	
37.	Write a program to Convert Infix expression to Postfix.	
38.	Write a program to Convert Infix expression to Prefix.	
39.	Write a program to implement Queue using Arrays.	
40.	Write a program to implement Queue using Linked List.	
41.	Write a program to implement Circular Queue using Arrays.	
42.	Write a program to implement Circular Queue using Linked List.	
43.	Write a program to implement Queue using Stack (Multi-Stack).	
44.	Write a program to implement Stack using Queue (Multi-Queue).	
45.	Write a program to create a Binary Tree along with Traversal (INORDER,PREF-ORDER,POST-ORDER)	
46.	WAP to implement following operations in BST: Insertion, Deletion, Traverse-(Inorder,Postorder,Preorder)	
47.	WAP to implement following operations on AVL trees: Insertion, Deletion, Traversal(Inorder,Preorder,Postorder)	
48.	WAP to store the graph information in form of adjacency matrix.	
49.	WAP to store the graph information in form of adjacency list.	

Question 1-7:

A menu driven program in C to:

- Implement insertion in an array.
- Implement deletion in an array.
- Find largest and 2nd largest element in array.
- Implement Linear Search.
- Implement Linear Search with multiple duplicate keys.
- Implement binary search.
- Implement binary search with recursion.

Code:

```
#include <stdio.h>

void printarr(int arr[],int n){
    printf("\nArray is: ");
    for(int i=0;i<n;i++)
        printf("%d ",arr[i]);
}

void insert(int el,int pos,int n,int arr[]){
    for(int i=n-1;i>=pos;i--)
        arr[i+1]=arr[i];
    arr[pos]=el;
}

void largest(int arr[],int n,int max,int sl){
    max=arr[0];
    for(int i=1;i<n;i++){
        if(arr[i]>max){
            sl=max;
            max=arr[i];
        }
    }
}
```

```

    }
    else if (arr[i]>sl && arr[i]<max){
        sl=arr[i];
    }
}
printf("\nLargest element of array is: %d",max);
printf("\nSecond Largest element of array is: %d",sl);
}

void linear(int arr[],int n, int ser_el){
    int i,count=0;
    for(i=0;i<n;i++){
        if(arr[i]==ser_el){
            printf("\nElement found at index: %d",i);
            count++;
        }
    }
    if(count==0)
        printf("\nElement Not Found :-(");
    else
        printf("\nElement's count: %d",count);
}

void binary(int arr[],int ser,int lb,int ub){
    while(lb<=ub){
        int mid=(lb+ub)/2;
        if(arr[mid]==ser){
            printf("\nElement found at index: %d",mid);
            break;
        }
    }
}

```

```

else if (ser > arr[mid])
    lb = mid + 1;

else if (ser < arr[mid]){
    ub = mid - 1;
}
}

if (lb > ub)
    printf("\nElement Not Found...");
}

int binary_rec(int arr[], int ser, int low, int high){
    if (low <= high){
        int mid = (low + high) / 2;
        if (arr[mid] == ser)
            return mid;
        else if (ser < arr[mid])
            return binary_rec(arr, ser, low, mid - 1);

        else if (ser > arr[mid])
            return binary_rec(arr, ser, mid + 1, high);
    }
    return -1;
}

void demlete(int pos, int n, int arr[]){
    for (int i = pos - 1; i < n - 1; i++)
        arr[i] = arr[i + 1];
}

```

```

int main() {
    int arr[10];
    int n,choice,ser_el,ser;
    int max,sl,el,pos;
    printf("Enter number of elements you want in array: ");
    scanf("%d",&n);

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

    printarr(arr,n);

    while(1){
        printf("\n\n1.Insert an element");
        printf("\n2.Largest and second largest element of the array");
        printf("\n3.Search using Linear Search");
        printf("\n4.Search using Binary Search");
        printf("\n5.Delete an element");
        printf("\n6.Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);

        if(choice==1){
            printf("\nEnter element which you want to insert: ");
            // int el=scanf("%d",&el);
            scanf("%d",&el);
            printf("\nEnter position at which element is to be inserted: ");

```



```

scanf("%d",&pos);
insert(el,pos,n,arr);
n++;
printf("\nElemented inserted sucessfully :-");
printarr(arr,n);
}
else if(choice==2){
    largest(arr, n, max,0);
}
else if(choice==3){
    printf("Enter element to search: ");
    scanf("%d",&ser_el);
    linear(arr,n,ser_el);
}
else if(choice==4){
    printf("Enter element to search: ");
    scanf("%d",&ser);
    printf("\n----Binary Search without Recursion----");
    binary(arr,ser,0,n-1);
    printf("\n----Binary Search with Recursion----");
    int res=binary_rec(arr,ser,0,n-1);
    if(res==-1)
        printf("\nElement not Found...");
    else
        printf("\nElement Found at index: %d",res);
}

```

```

else if (choice==5){
    int pos;
    printf("\nEnter position of the element to be deleted: ");
    scanf("%d",&pos);
    if(pos>=1 && pos<=n){
        demlete(pos,n,arr);
        n--;
        printf("\nElemented deleted sucessfully :-");
        printarr(arr,n);
    }
    else
        printf("Wrong position entered, Please Enter Correct postion....");
}
else if(choice==6){
    printf("Program Terminated...\n");
    break;
}
}
return 0;
}

```

Outputs:

```
5 ./math
Enter number of elements you want in array: 5
Enter elements of the array:
1
3
3
4
7

Array is: 1 3 3 4 7
```

```
1.Insert an element
2.Largest and second largest element of the array
3.Search using Linear Search
4.Search using Binary Search
5.Delete an element
6.Exit
Enter your choice: 1

Enter element which you want to insert: 2

Enter position at which element is to be inserted: 1

Elemented inserted sucessfully :-)
Array is: 1 2 3 3 4 7
```

```
1.Insert an element
2.Largest and second largest element of the array
3.Search using Linear Search
4.Search using Binary Search
5.Delete an element
6.Exit
Enter your choice: 2

Largest element of array is: 7
Second Largest element of array is: 4
```

```
1.Insert an element
2.Largest and second largest element of the array
3.Search using Linear Search
4.Search using Binary Search
5.Delete an element
6.Exit
Enter your choice: 3
Enter element to search: 3

Element found at index: 2
Element found at index: 3
Element's count: 2
```

```
1.Insert an element
2.Largest and second largest element of the array
3.Search using Linear Search
4.Search using Binary Search
5.Delete an element
6.Exit
Enter your choice: 4
Enter element to search: 4

----Binary Search without Recursion----
Element found at index: 4
----Binary Search with Recursion----
Element Found at index: 4
```

```
1.Insert an element
2.Largest and second largest element of the array
3.Search using Linear Search
4.Search using Binary Search
5.Delete an element
6.Exit
Enter your choice: 5

Enter position of the element to be deleted: 3

Elemented deleted sucessfully :-)
Array is: 1 2 3 4 7
```

Question 8-12:

A menu driven program in C to implement:

- Bubble Sort
- Insertion Sort
- Selection Sort
- Count Sort
- Radix Sort

Code:

```
#include <stdio.h>

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

void insertion_sort(int arr[],int n){
    int temp,j;
    for(int i=1;i<n;i++){
        temp=arr[i];
        j=i-1;
        while(j>=0 && arr[j]>temp){
            arr[j+1]=arr[j];
            j--;
        }
        arr[j+1]=temp;
    }
}
```

```

void bubble_sort(int arr[],int n){
    for(int i=0;i<n-1;i++){
        int flag=0;
        for(int j=0;j<n-i-1;j++){
            if(arr[j]>arr[j+1]){
                int temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
                flag=1;
            }
            if(flag==0)
                break;
        }
    }
}

void selection_sort(int arr[],int n){
    int min;
    for(int i=1;i<n;i++){
        min=i;
        for(int j=i+1;j<n;j++){
            if(arr[j]<arr[min])
                min=j;
        }
        if(min!=i)
        {
            int temp=arr[i];
            arr[i]=arr[min];
            arr[min]=temp;
        }
    }
}

```

```

    }
}
}
void count_sort(int arr[],int n){
    int max=arr[0];
    int output[n];
    for(int i=1;i<n;i++){
        if(arr[i]>max)
            max=arr[i];
    }
    int count[max+1];
    for(int i=0;i<max+1;i++)
        count[i]=0;

    for(int i=0;i<n;i++)
        count[arr[i]]++;

    for(int i=1;i<=max;i++)
        count[i]=count[i]+count[i-1];

    for(int i=n-1;i>=0;i--){
        output[count[arr[i]]-1]=arr[i];
        count[arr[i]]--;
    }
    for(int i=0;i<n;i++)
        arr[i]=output[i];
}

```

```

void radix_count_sort(int arr[],int n,int pos){
    int output[n];
    int count[10]={0};
    for(int i=0;i<n;i++)
        count[(arr[i]/pos)%10]++;

    for(int i=1;i<=9;i++)
        count[i]+=count[i-1];

    for(int i=n-1;i>=0;i--){
        output[count[(arr[i]/pos)%10]-1]=arr[i];
        count[(arr[i]/pos)%10]--;
    }
    for(int i=0;i<n;i++)
        arr[i]=output[i];
}

void radix_sort(int arr[],int n){
    int max=arr[0];
    for(int i=1;i<n;i++){
        if(arr[i]>max)
            max=arr[i];
    }
    for(int pos=1;max/pos>0;pos=pos*10)
        radix_count_sort(arr,n,pos);
}

```



```

int main() {
    int arr[10];
    int n, choice;
    printf("Enter number of elements of array: ");
    scanf("%d",&n);
    printf("\nEnter elements of the array: \n");
    for(int i=0;i<n;i++)
        scanf("%d",&arr[i]);

    while(1){
        printf("\n\n1.Sort Array using Insertion Sort");
        printf("\n2.Sort Array using Bubble Sort");
        printf("\n3.Sort Array using Selection Sort");
        printf("\n4.Sort Array using Count Sort");
        printf("\n5.Sort Array using Radix Sort");
        printf("\n6.Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);

        if (choice==1){
            insertion_sort(arr,n);
            printf("\nSorted Array using Insertion Sort: ");
            printarray(arr,n);
        }
        else if(choice==2){
            bubble_sort(arr,n);
            printf("\nSorted Array using Bubble Sort: ");
            printarray(arr,n);

```

```

    }
    else if(choice==3){
        selection_sort(arr,n);
        printf("\nSorted Array using Selection Sort: ");
        printarray(arr,n);
    }
    else if(choice==4){
        count_sort(arr,n);
        printf("\nSorted Array using Count Sort: ");
        printarray(arr,n);
    }
    else if(choice==5){
        radix_sort(arr,n);
        printf("\nSorted Array using Radix Sort: ");
        printarray(arr,n);
    }
    else if(choice==6){
        printf("Program Terminated...");
        break;
    }
}
return 0;
}

```

Outputs:

```
➤ ./main
Enter number of elements of array: 7

Enter elements of the array:
5
2
334
123
67
7
1

Array is: 5 2 334 123 67 7 1
```

1. Insertion Sort

```
1.Sort Array using Insertion Sort
2.Sort Array using Bubble Sort
3.Sort Array using Selection Sort
4.Sort Array using Count Sort
5.Sort Array using Radix Sort
6.Exit
Enter your choice: 1

Sorted Array using Insertion Sort: 1 2 5 7 67 123 334
```

2. Bubble Sort

```
1.Sort Array using Insertion Sort
2.Sort Array using Bubble Sort
3.Sort Array using Selection Sort
4.Sort Array using Count Sort
5.Sort Array using Radix Sort
6.Exit
Enter your choice: 2

Sorted Array using Bubble Sort: 1 2 5 7 67 123 334
```

3. Selection Sort

```
1.Sort Array using Insertion Sort
2.Sort Array using Bubble Sort
3.Sort Array using Selection Sort
4.Sort Array using Count Sort
5.Sort Array using Radix Sort
6.Exit
Enter your choice: 3

Sorted Array using Selection Sort: 1 2 5 7 67 123 334
```

4. Count Sort

```
1.Sort Array using Insertion Sort
2.Sort Array using Bubble Sort
3.Sort Array using Selection Sort
4.Sort Array using Count Sort
5.Sort Array using Radix Sort
6.Exit
Enter your choice: 4

Sorted Array using Count Sort: 1 2 5 7 67 123 334
```

5. Radix Sort

```
1.Sort Array using Insertion Sort
2.Sort Array using Bubble Sort
3.Sort Array using Selection Sort
4.Sort Array using Count Sort
5.Sort Array using Radix Sort
6.Exit
Enter your choice: 5

Sorted Array using Radix Sort: 1 2 5 7 67 123 334
```

Question 13-14:

A menu driven program in C to:

- Add two matrices
- Subtract two matrices
- Multiply two matrices
- Check two matrices are identical or not

Code:

```
#include <stdio.h>

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

void enter_mat(int n,int m,int **arr){
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &arr[i][j]);
}

int main(void) {
    int n1, m1,n2,m2, choice;
    printf("Enter number of rows of 1st matrix: ");
    scanf("%d", &n1);
    printf("\nEnter number of columns of 1st matrix: ");
    scanf("%d", &m1);
```

```

int **mat1=(int **)malloc(n1*sizeof(int **));
for(int i=0;i<n1;i++)
    mat1[i]=(int *)malloc(m1*sizeof(int));

printf("\nEnter elements of 1st matrix:\n");
enter_mat(n1, m1, mat1);

printf("\nEnter number of rows of 2nd matrix: ");
scanf("%d", &n2);
printf("\nEnter number of columns of 2nd matrix: ");
scanf("%d", &m2);

int **mat2=(int **)malloc(n2*sizeof(int **));
for(int i=0;i<n2;i++)
    mat2[i]=(int *)malloc(m2*sizeof(int));

printf("\nEnter elements of 2nd matrix:\n");
enter_mat(n2, m2, mat2);

printf("\n1st Matrix is:\n");
print_matrix(n1, m1, mat1);
printf("\n2nd Matrix is:\n");
print_matrix(n2, m2, mat2);

int **res=(int **)malloc(n1*sizeof(int **));
for(int i=0;i<n1;i++)
    res[i]=(int *)malloc(m2*sizeof(int));

```

```

while (1) {
    printf("\n\n1.Add two Matrices");
    printf("\n2.Subtract two Matrices");
    printf("\n3.Multiply two Matrices");
    printf("\n4.Check if two Matrices are Identical or not");
    printf("\n5.Exit");
    printf("\nEnter your choice: ");
    scanf("%d", &choice);
    if(choice==1){
        if(n1==n2 && m1==m2){
            for(int i=0;i<n1;i++)
                for(int j=0;j<m1;j++)
                    res[i][j]=mat1[i][j]+mat2[i][j];

            printf("\nAddition is: \n");
            print_matrix(n1,m1,res);
        }
        else
            printf("\nCannot perform Addition as order of both matrices is not
same...");
    }
    else if (choice==2){
        if(n1==n2 && m1==m2){
            for(int i=0;i<n1;i++)
                for(int j=0;j<m1;j++)
                    res[i][j]=mat1[i][j]-mat2[i][j];

            printf("\nSubtraction is: \n");

```

```

    print_matrix(n1,m1,res);
}
else
    printf("\nCannot perform Addition as order of both matrices is not
same...");
}
else if (choice==3){
if(m1==n2){
    for(int i=0;i<n1;i++){
        for(int j=0;j<m2;j++){
            res[i][j]=0;
            for(int k=0;k<n2;k++)
                res[i][j]=res[i][j] + mat1[i][k] * mat2[k][j];
        }
    }
    printf("\nMultiplication is: \n");
    print_matrix(n1,m2,res);
}
else
    printf("Cannot Perform Matrix Multiplication...");
}
else if (choice==4){
    int flag=1;
    if(n1==n2 && m1==m2){
        for(int i=0;i<n1;i++){
            for(int j=0;j<m1;j++){
                if(mat1[i][j]!=mat2[i][j]){
                    flag=0;

```



```
        break;
    }
}
}
}
else{
    flag=0;
}
if(flag==0)
    printf("The two matrices are not identical :-(");
else
    printf("The two matrices are identical :-)");
}
else if (choice==5){
    printf("\nProgram Terminated...");
    break;
}
}
return 0;
}
```

Output:

```
➤ ./main
Enter number of rows of 1st matrix: 3
Enter number of columns of 1st matrix: 3
Enter elements of 1st matrix:
1
2
3
4
5
6
7
8
9

Enter number of rows of 2nd matrix: 3
Enter number of columns of 2nd matrix: 3
```

```
Enter elements of 2nd matrix:
8
5
6
4
3
2
1
0
8
```

```
1st Matrix is:
1 2 3
4 5 6
7 8 9
```

```
2nd Matrix is:
8 5 6
4 3 2
1 0 8
```

```
1.Add two Matrices
2.Subtract two Matrices
3.Multiply two Matrices
4.Check if two Matrices are Identical or not
5.Exit
Enter your choice: 1
```

Addition is:

```
9   7   9
8   8   8
8   8  17
```

```
1.Add two Matrices
2.Subtract two Matrices
3.Multiply two Matrices
4.Check if two Matrices are Identical or not
5.Exit
Enter your choice: 2
```

Subtraction is:

```
-7  -3  -3
0   2   4
6   8   1
```

```
1.Add two Matrices
2.Subtract two Matrices
3.Multiply two Matrices
4.Check if two Matrices are Identical or not
5.Exit
Enter your choice: 3
```

Multiplication is:

```
19  11  34
58  35  82
97  59 130
```

```
1.Add two Matrices
2.Subtract two Matrices
3.Multiply two Matrices
4.Check if two Matrices are Identical or not
5.Exit
Enter your choice: 4
The two matrices are not identical :-(
```

Question 15-17:

A menu driven program in C to:

- Check whether given matrix is identity or not.
- Check given matrix is sparse or not.
- Convert sparse matrix in row triplet form.

Code:

```
#include <stdio.h>

int main(void) {
    int n, m, choice;

    printf("Enter number of rows of matrix: ");
    scanf("%d", &n);

    printf("\nEnter number of columns of matrix: ");
    scanf("%d", &m);

    int **mat=(int **)malloc(n*sizeof(int *));
    for(int i=0;i<n;i++)
        mat[i]=(int *)malloc(m*sizeof(int));

    printf("\nEnter elements of matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &mat[i][j]);

    printf("Entered matrix is: \n");
    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
}
```

```

while (1) {
    printf("\n\n1.Check if Matrix is Identity or Not");
    printf("\n\n2.Check if Matrix is Sparse or Not");
    printf("\n\n3.Convert Sparse matrix into Row Triplet Form");
    printf("\n\n4.Exit");
    printf("\n\nEnter your choice: ");
    scanf("%d", &choice);
    if(choice==1){
        int flag=0;
        for(int i=0;i<n;i++)
            for(int j=0;j<m;j++)
                if((i==j && mat[i][j]!=1) || (i!=j && mat[i][j]!=0))
                    flag=1;

        if(flag==1)
            printf("\nMatrix is not Identity :(");
        else
            printf("\nMatrix is Identity :)");
    }
    else if(choice==2){
        int threshold=n*m;
        int count=0;
        for(int i=0;i<n;i++)
            for(int j=0;j<m;j++)
                if(mat[i][j]==0)
                    count++;

        if(count>threshold/2)
            printf("\nEntered Matrix is Sparse :-");
        else

```

```

        printf("\nEntered Matrix is not Sparse :-(");
    }
else if(choice==3){
    int rt[10][3],count=0,k=0;
    printf("\nRow Triplet Form: \n");
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(mat[i][j]!=0){
                count++;
                rt[k][0]=i;
                rt[k][1]=j;
                rt[k][2]=mat[i][j];
                k++;
            }
        }
    }
    for(int i=0;i<count;i++)
        printf("(%d, %d, %d)\n", rt[i][0], rt[i][1], rt[i][2]);
}
else if(choice==4){
    printf("Program Terminated...");
    break;
}
}
return 0;
}

```

Output:

```
➤ ./main
Enter number of rows of matrix: 3
Enter number of columns of matrix: 3
Enter elements of matrix:
1
0
0
0
1
0
0
0

1
Entered matrix is:
1 0 0
0 1 0
0 0 1
```

1. Identity

```
1.Check if Matrix is Identity or Not
2.Check if Matrix is Sparse or Not
3.Convert Sparse matrix into Row Triplet Form
4.Exit
Enter your choice: 1

Matrix is Identity :)
```

2. Sparse

```
1.Check if Matrix is Identity or Not
2.Check if Matrix is Sparse or Not
3.Convert Sparse matrix into Row Triplet Form
4.Exit
Enter your choice: 2

Entered Matrix is Sparse :-)
```

3. Row Triplet Form

```
1.Check if Matrix is Identity or Not
2.Check if Matrix is Sparse or Not
3.Convert Sparse matrix into Row Triplet Form
4.Exit
Enter your choice: 3

Row Triplet Form:
(0, 0, 1)
(1, 1, 1)
(2, 2, 1)
```


Question 18-20:

A menu driven program in C to:

- Print all diagonal value of matrix.
- Print sum of all rows in matrix.
- Sort all rows in ascending order.

Code:

```
#include <stdio.h>

void sort(int *mat,int n){
    int temp,j;
    for(int i=1;i<n;i++){
        temp=mat[i];
        j=i-1;
        while(j>=0 && mat[j]>temp){
            mat[j+1]=mat[j];
            j--;
        }
        mat[j+1]=temp;
    }
}

int main(void) {
    int n, m,choice;
    printf("Enter number of rows of matrix: ");
    scanf("%d", &n);
    printf("\nEnter number of columns of matrix: ");
    scanf("%d", &m);

    int **mat=(int **)malloc(n*sizeof(int *));
    for(int i=0;i<n;i++)
```

```

    mat[i]=(int *)malloc(m*sizeof(int));

printf("\nEnter elements of matrix:\n");
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        scanf("%d", &mat[i][j]);

printf("Entered matrix is: \n");
for (int i = 0; i < n; i++){
    for (int j = 0; j < m; j++)
        printf("%d ", mat[i][j]);
    printf("\n");
}

while (1) {
    printf("\n\n1.Print all Diagonal Elements");
    printf("\n2.Print Sum of all Rows");
    printf("\n3.Arrange all Rows in ascending order");
    printf("\n4.Exit");
    printf("\nEnter your choice: ");
    scanf("%d", &choice);

    if(choice==1){
        printf("\nMain Diagonal elements are: ");
        for(int i=0;i<n;i++)
            for(int j=0;j<m;j++)
                if(i==j)
                    printf("%d ",mat[i][j]);

        printf("\nRight Diagonal elements are: ");
        for(int i=0;i<n;i++)

```

```

        for(int j=0;j<m;j++)
            if(i+j==n-1)
                printf("%d ",mat[i][j]);
    }

    else if(choice==2){
        int sum=0;
        for(int i=0;i<n;i++)
            for(int j=0;j<m;j++)
                sum=sum+mat[i][j];

        printf("\nSum of all rows: %d",sum);
    }

    else if(choice==3){
        for(int i=0;i<n;i++)
            sort(mat[i],n);

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

        else if(choice==4){
            printf("Program Terminated...");
            break;
        }
    }

    return 0;
}

```

Output:

```
➤ ./main
Enter number of rows of matrix: 3
Enter number of columns of matrix: 3
Enter elements of matrix:
6
3
1
9
0
5
3
5
2
Entered matrix is:
6 3 1
9 0 5
3 5 2
```

1. Diagonal Elements

```
1.Print all Diagonal Elements
2.Print Sum of all Rows
3.Arrange all Rows in ascending order
4.Exit
Enter your choice: 1

Main Diagonal elements are: 1 5 5
Right Diagonal elements are: 6 5 2
```

2. Sum of All Rows

```
1.Print all Diagonal Elements
2.Print Sum of all Rows
3.Arrange all Rows in ascending order
4.Exit
Enter your choice: 2

Sum of all rows: 34
```

3. Rows in Ascending Order

```
1.Print all Diagonal Elements
2.Print Sum of all Rows
3.Arrange all Rows in ascending order
4.Exit
Enter your choice: 3
1 3 6
0 5 9
2 3 5
```

Question 21:

Write a program to find missing element in an array.

```
#include <stdio.h>

int main(void) {
    int n,sum=0;
    printf("Enter number of elements of array: ");
    scanf("%d",&n);
    int *arr=(int *)malloc(n*sizeof(int));
    printf("Enter elements of array: ");
    for(int i=0;i<n;i++)
        scanf("%d",arr+i);
    printf("\nEntered Array is: ");
    for(int i=0;i<n;i++)
        printf("%d ",*(arr+i));

    int org_sum=(n*(n+1))/2;
    for(int i=0;i<n;i++)
        sum=sum + *(arr+i);

    int res=org_sum-sum;
    if(res!=0)
        printf("\nMissing element in array is: %d",res);
    else
        printf("\nThere is no missing element in array..");
    return 0;
}
```

Output:

```
> ./main
Enter number of elements of array: 5
Enter elements of array: 1
2
4
5
0

Entered Array is: 1 2 4 5 0
Missing element in array is: 3
```

```
> ./main
Enter number of elements of array: 4
Enter elements of array: 1
2
3
4

Entered Array is: 1 2 3 4
There is no missing element in array..
```

Question 22:

Write a program to remove duplicates from an array.

```
#include <stdio.h>

void selection_sort(int *arr,int n){
    int min;
    for(int i=0;i<n-1;i++){
        min=i;
        for(int j=i+1;j<n;j++){
            if(arr[j]<arr[min])
                min=j;
        }
        if(min!=i){
            int temp=arr[min];
            arr[min]=arr[i];
            arr[i]=temp;
        }
    }
}

int remove_dup(int *arr,int n){
    int i=0,j=1;
    while(j<n){
        if(arr[i]==arr[j])
            j++;
        else{
            i++;
            arr[i]=arr[j];
        }
    }
    return i;
}
```



```

int main(void) {
    int n;
    printf("Enter number of elements of array: ");
    scanf("%d",&n);
    int *arr=(int *)malloc(n*sizeof(int));
    printf("Enter elements of array: ");
    for(int i=0;i<n;i++)
        scanf("%d",arr+i);
    printf("\nArray is: ");
    for(int i=0;i<n;i++)
        printf("%d ",*(arr+i));
    selection_sort(arr,n);
    printf("\nArray after Sorting is: ");
    for(int i=0;i<n;i++)
        printf("%d ",*(arr+i));
    int val=remove_dup(arr, n);
    printf("\nArray after removing duplicates is: ");
    for(int i=0;i<=val;i++)
        printf("%d ",*(arr+i));
}

```

Output:

```

$ ./math
Enter number of elements of array: 7
Enter elements of array: 3
2
1
2
1
3
4

Array is: 3 2 1 2 1 3 4
Array after Sorting is: 1 1 2 2 3 3 4
Array after removing duplicates is: 1 2 3 4

```

Question 23:

Write a program to print duplicate elements in an array.

Code:

```
#include <stdio.h>

int main(void) {
    int count[9]={0},n;
    printf("Enter size of array: ");
    scanf("%d",&n);
    int *arr=(int*)malloc(n*sizeof(int));
    printf("Enter elements of array:\n");
    for(int i=0;i<n;i++)
        scanf("%d",arr+i);

    printf("\nArray is: ");
    for(int i=0;i<n;i++){
        printf("%d ",*(arr+i));
        count[arr[i]]++;
    }
    printf("\nDuplicate elements are: ");
    for(int i=0;i<n;i++){
        if(count[i]>1)
            printf("%d ",i);
    }
    return 0;
}
```

Output:

```
➤ ./math
Enter size of array: 7
Enter elements of array:
1
2
2
2
3
3
4

Array is: 1 2 2 2 3 3 4
Duplicate elements are: 2 3 ➤ □
```

Question 24:

Write a program to find sum of elements of an array using malloc(), calloc(), realloc() and free() functions.

Code:

```
#include <stdio.h>

int sumOfArray(int *arr,int n){
    int sum=0;
    for(int i=0;i<n;i++)
        sum+=*(arr+i);
    return sum;
}

void enter_array(int *arr,int start,int n){
    printf("\nEnter elements of array:\n");
    for(int i=start;i<n;i++)
        scanf("%d",arr+i);
}

void display_array(int *arr,int n){
    for(int i=0;i<n;i++)
        printf("%d ",*(arr+i));
}

int main(void) {
    int n;
    int *p;
    printf("Enter size of array: ");
    scanf("%d",&n);
    p=(int *)malloc(n*sizeof(int));
    enter_array(p, 0,n);
    printf("Array is: ");
    display_array(p,n);
}
```

```

int sum=sumOfArray(p, n);
printf("\nSum of elements using malloc(): %d",sum);

free(p);
printf("\n\nArray after using free() is: ");
display_array(p, n);

p=(int *)calloc(n,sizeof(int));
printf("\n");
enter_array(p, 0,n);
printf("Array is: ");
display_array(p,n);
sum=sumOfArray(p, n);
printf("\nSum of elements using calloc(): %d",sum);

p=realloc(p,2*n*sizeof(int));
printf("\n\nArray after realloc(array,2*size: ");
display_array(p, 2*n);
printf("\nEnter remaining elements of array...");
enter_array(p, n, 2*n);
printf("Array is: ");
display_array(p,2*n);

sum=sumOfArray(p, 2*n);
printf("\nSum of elements after using realloc(): %d",sum);

return 0;
}

```

Output:

```
➤ ./main
Enter size of array: 4

Enter elements of array:
1
2
3
4
Array is: 1 2 3 4
Sum of elements using malloc(): 10

Array after using free() is: 7778 0 2055507642 422874380

Enter elements of array:
7
3
5
9
Array is: 7 3 5 9
Sum of elements using calloc(): 24
```

```
Array after realloc(array,2*size): 7 3 5 9 0 0 132369 0
Enter remaining elements of array...
Enter elements of array:
4
2
8
1
Array is: 7 3 5 9 4 2 8 1
Sum of elements after using realloc(): 39➤
```

Question 25:

Write a program to create array of structures of 5 employees with their EmpId, Ename and Salary. Display all details of the employee with highest salary.

Code:

```
#include <stdio.h>

struct emp{
int eid;
char ename[20];
int esal;
} s;

int main(void) {
    int max=0, val;
    struct emp s[5];
    for(int i=0; i<5; i++){
        printf("Enter Employee ID: ");
        scanf("%d", &s[i].eid);
        printf("Enter Employee Name: ");
        scanf("%s", s[i].ename);
        printf("Enter Employee Salary: ");
        scanf("%d", &s[i].esal);
        printf("\n");
    }
    for(int i=0; i<5; i++){
        if(s[i].esal > max){
            max=s[i].esal;
            val=i;
        }
    }
}
```

```
printf("\n---Employee with highest salary---\n");  
printf("Emp ID: %d\nEmp Name: %s\nEmp Salary: %d\n",s[val].eid,s[val].ename,max);  
return 0;  
}
```

Output:

```
➤ ./main  
Enter Employee ID: 1  
Enter Employee Name: Hitesh  
Enter Employee Salary: 50000  
  
Enter Employee ID: 2  
Enter Employee Name: Sajal  
Enter Employee Salary: 35000  
  
Enter Employee ID: 3  
Enter Employee Name: Bhavya  
Enter Employee Salary: 45000  
  
Enter Employee ID: 4  
Enter Employee Name: Rishi  
Enter Employee Salary: 51000  
  
Enter Employee ID: 5  
Enter Employee Name: Vedant  
Enter Employee Salary: 6000
```

```
----Employee with highest salary----  
Emp ID: 4  
Emp Name: Rishi  
Emp Salary: 51000  
➤ █
```


Question 26:

Write a program to perform following operations on singly linked list:- Create, Traverse, Insertion and Deletion(beg, end , specific), Length of list, Searching, Reverse list.

Code:

```
#include <stdio.h>

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

struct node *start=NULL;
struct node *newNode,*temp;

void allocate_memory(){
    newNode=(struct node*)malloc(sizeof(struct node));
    printf("\nEnter data of node: ");
    scanf("%d",&newNode->data);
    newNode->next=NULL;
}

void node_insert_end(){
    allocate_memory();
    if(start==NULL)
        start=newNode;
    else{
        temp=start;
        while(temp->next!=NULL){
            temp=temp->next;
        }
        temp->next=newNode;
    }
}
```

```

    }
}
void node_insert_beg(){
    allocate_memory();
    if(start==NULL)
        start=newNode;
    else{
        newNode->next=start;
        start=newNode;
    }
}
void node_insert_specific(){
    int i=1,pos;
    temp=start;
    allocate_memory();
    if(start==NULL)
        start=newNode;

    else{
        printf("Enter Position where you want to insert node: ");
        scanf("%d",&pos);
        if(pos==1){
            newNode->next=start;
            start=newNode;
        }
        else{
            while(i<pos-1){
                temp=temp->next;
                i++;
            }
            newNode->next=temp->next;

```

```

        temp->next=newNode;
    }
}

void node_delete_end(){
    struct node *prev;
    temp=start;
    while(temp->next!=NULL){
        prev=temp;
        temp=temp->next;
    }
    prev->next=NULL;
    free(temp);
}

void node_delete_beg(){
    temp=start;
    start=temp->next;
    free(temp);
}

void node_delete_specific(){
    int i=1,pos;
    struct node *var;
    temp=start;
    printf("Enter Position of node to delete: ");
    scanf("%d",&pos);
    if(pos==1){
        start=temp->next;
        free(temp);
    }
    else{

```

```

while(i<pos-1){
    temp=temp->next;
    i++;
}
var=temp->next;
temp->next=var->next;
free(var);
}
}

void print_list(){
    struct node *temp2=start;
    printf("\nData of Linked List: ");
    while(temp2!=NULL){
        printf("%d-->",temp2->data);
        temp2=temp2->next;
    }
}

void length(){
    temp=start;
    int count=0;
    while(temp!=NULL){
        count++;
        temp=temp->next;
    }
    printf("Length of List is: %d",count);
}

void reverse_list(){
    struct node *prev=NULL,*curr=start,*nxt;
    while(curr!=NULL){
        nxt=curr->next;
        curr->next=prev;
    }
}

```

```

    prev=curr;
    curr=nxt;
}
start=prev;
}
int main(void) {
    int choice;
    while(1){
        printf("\n\n1.Insert New Node at end");
        printf("\n2.Insert New Node at begining");
        printf("\n3.Insert New Node at specific pos");
        printf("\n4.Delete Node from end");
        printf("\n5.Delete Node from begining");
        printf("\n6.Delete Node from specific pos");
        printf("\n7.Length of List");
        printf("\n8.Reverse a List");
        printf("\n9.Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);
        if(choice==1){
            node_insert_end();
            printf("\nNode Inserted Successfully :");
            print_list();
        }
        if(choice==2){
            node_insert_beg();
            printf("\nNode Inserted Successfully :");
            print_list();
        }
        if(choice==3){
            node_insert_specific();

```

```

    printf("\nNode Inserted Successfully :");
    print_list();
}
if(choice==4){
    node_delete_end();
    printf("\nNode Deleted Successfully :");
    print_list();
}
if(choice==5){
    node_delete_beg();
    printf("\nNode Deleted Successfully :");
    print_list();
}
if(choice==6){
    node_delete_specific();
    printf("\nNode Deleted Successfully :");
    print_list();
}
if(choice==7){
    length(); }
if(choice==8){
    reverse_list();
    print_list();
}
if(choice==9){
    printf("Program Terminated...");
    break;
}
}
return 0;
}

```

Output:

```
Node Inserted Successfully :)
Data of Linked List: 1-->

1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Reverse a List
9.Exit
Enter your choice: 1
```

```
Enter data of node: 2
```

```
Node Inserted Successfully :)
Data of Linked List: 1-->2-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Reverse a List
9.Exit
Enter your choice: 3
```

```
Enter data of node: 4
```

```
Enter Position where you want to insert node: 2
```

```
Node Inserted Successfully :)
Data of Linked List: 3-->4-->1-->2-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Reverse a List
9.Exit
Enter your choice: 6
Enter Position of node to delete: 3

Node Deleted Successfully :)
Data of Linked List: 3-->4-->2-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Reverse a List
9.Exit
Enter your choice: 4

Node Deleted Successfully :)
Data of Linked List: 3-->4-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Reverse a List
9.Exit
Enter your choice: 8

Data of Linked List: 5-->3-->2-->4-->
```


Question 27:

Write a program to perform following operations on doubly linked list:- Create, Traverse, Insertion and Deletion(beg, end , specific), Length of list, Reverse list.

Code:

```
#include <stdio.h>

struct node{
    int data;
    struct node *prev;
    struct node *next;
};

struct node *newNode,*temp,*start=NULL;

void allocate_memory(){
    newNode=(struct node*)malloc(sizeof(struct node));
    printf("\nEnter data of node: ");
    scanf("%d",&newNode->data);
    newNode->next=NULL;
    newNode->prev=NULL;
}

int length_of_list(){
    temp=start;
    int count=0;
    while(temp!=NULL){
        temp=temp->next;
        count++;
    }
    return count;
}

void print_list(){
```

```

temp=start;
if(temp==NULL)
    printf("\nList is empty!!");
else{
    printf("\nList is: ");
    while(temp!=NULL){
        printf("<--%d-->",temp->data);
        temp=temp->next;
    }
}

void node_insert_beg(){
    allocate_memory();
    if(start==NULL)
        start=newNode;
    else{
        newNode->next=start;
        start->prev=newNode;
        start=newNode;
    }
}

void node_insert_end(){
    temp=start;
    allocate_memory();
    if(start==NULL)
        start=newNode;
    else{
        while(temp->next!=NULL)
            temp=temp->next;
        newNode->prev=temp;
        temp->next=newNode;
    }
}

```

```

    }
}
void node_insert_specific(){
    int i=1,pos;
    temp=start;
    allocate_memory();
    if(start==NULL)
        start=newNode;
    else{
        printf("Enter position where you want to insert: ");
        scanf("%d",&pos);
        if(pos==1){
            newNode->next=start;
            start->prev=newNode;
            start=newNode;
        }
        else{
            while(i<pos-1){
                temp=temp->next;
                i++;
            }
            newNode->next=temp->next;
            newNode->prev=temp;
            temp->next->prev=newNode;
            temp->next=newNode;
        }
    }
}
void node_delete_end(){
    struct node *prev2;
    int length=length_of_list();

```

```

temp=start;
if(length==1){
    free(temp);
    start=NULL;
}
else{
    while(temp->next!=NULL){
        prev2=temp;
        temp=temp->next;
    }
    prev2->next=NULL;
    free(temp);
}
}

void node_delete_beg(){
    int length=length_of_list();
    temp=start;
    if(length==1){
        free(temp);
        start=NULL;
    }
    else{
        temp->next->prev=NULL;
        start=temp->next;
        free(temp);
    }
}

void node_delete_specific(){
    int i=1,pos;
    struct node *var;
    temp=start;

```

```

printf("Enter Position of node to delete: ");
scanf("%d",&pos);
if(pos==1){
    start=temp->next;
    temp->next->prev=NULL;
    free(temp);
}
else{
    while(i<pos){
        temp=temp->next;
        i++;
    }
    temp->prev->next=temp->next;
    temp->next->prev=temp->prev;
    free(temp);
}
}

void reverse_list(){
    temp=start;
    while(temp->next!=NULL)
        temp=temp->next;

    printf("Reversed List is: ");
    while(temp!=NULL){
        printf("<--%d-->",temp->data);
        temp=temp->prev;
    }
}

int main(void) {
    int choice;
    while(1){

```

```

printf("\n\n1.Insert node at end");
printf("\n\n2.Insert node at beg");
printf("\n\n3.Insert node at specific pos");
printf("\n\n4.Delete node at end");
printf("\n\n5.Delete node at beg");
printf("\n\n6.Delete node at specific pos");
printf("\n\n7.Reverse List");
printf("\n\n8.Length of List");
printf("\n\n9.Exit");
printf("\n\nEnter your choice: ");
scanf("%d",&choice);

if(choice==1){
    node_insert_end();
    printf("\n\nNode Inserted Successfully :)");
    print_list();
}
if(choice==2){
    node_insert_beg();
    printf("\n\nNode Inserted Successfully :)");
    print_list();
}
if(choice==3){
    node_insert_specific();
    printf("\n\nNode Inserted Successfully :)");
    print_list();
}
if(choice==4){
    node_delete_end();
    printf("\n\nNode Deleted Successfully :)");
    print_list();
}

```

```

    }
    if(choice==5){
        node_delete_beg();
        printf("\nNode Deleted Successfully :");
        print_list();
    }
    if(choice==6){
        node_delete_specific();
        printf("\nNode Deleted Successfully :");
        print_list();
    }
    if(choice==7){
        reverse_list();
    }
    if(choice==8){
        int len=length_of_list();
        printf("Length of List is: %d",len);
    }
    if(choice==9){
        printf("Program Terminated...");
        break;
    }
}
return 0;
}

```

Output:

```
1.Insert node at end
2.Insert node at beg
3.Insert node at specific pos
4.Delete node at end
5.Delete node at beg
6.Delete node at specific pos
7.Reverse List
8.Length of List
9.Exit
Enter your choice: 1

Enter data of node: 2

Node Inserted Successfully :)
List is: <--1--><--2-->
```

```
1.Insert node at end
2.Insert node at beg
3.Insert node at specific pos
4.Delete node at end
5.Delete node at beg
6.Delete node at specific pos
7.Reverse List
8.Length of List
9.Exit
Enter your choice: 2

Enter data of node: 3

Node Inserted Successfully :)
List is: <--3--><--1--><--2-->
```



```
1.Insert node at end
2.Insert node at beg
3.Insert node at specific pos
4.Delete node at end
5.Delete node at beg
6.Delete node at specific pos
7.Reverse List
8.Length of List
9.Exit
Enter your choice: 3

Enter data of node: 4
Enter position where you want to insert: 2

Node Inserted Successfully :)
List is: <--3--><--4--><--1--><--2-->
```

```
1.Insert node at end
2.Insert node at beg
3.Insert node at specific pos
4.Delete node at end
5.Delete node at beg
6.Delete node at specific pos
7.Reverse List
8.Length of List
9.Exit
Enter your choice: 4

Node Deleted Successfully :)
List is: <--3--><--4--><--1-->
```

```
1.Insert node at end
2.Insert node at beg
3.Insert node at specific pos
4.Delete node at end
5.Delete node at beg
6.Delete node at specific pos
7.Reverse List
8.Length of List
9.Exit
Enter your choice: 5

Node Deleted Successfully :)
List is: <--4--><--1-->
```

```
List is: <--4--><--1--><--2--><--3--><--6-->
```



- 1.Insert node at end
- 2.Insert node at beg
- 3.Insert node at specific pos
- 4.Delete node at end
- 5.Delete node at beg
- 6.Delete node at specific pos
- 7.Reverse List
- 8.Length of List
- 9.Exit

Enter your choice: 6

Enter Position of node to delete: 4

Node Deleted Successfully :)

```
List is: <--4--><--1--><--2--><--6-->
```

- 1.Insert node at end
- 2.Insert node at beg
- 3.Insert node at specific pos
- 4.Delete node at end
- 5.Delete node at beg
- 6.Delete node at specific pos
- 7.Reverse List
- 8.Length of List
- 9.Exit

Enter your choice: 7

```
Reversed List is: <--6--><--2--><--1--><--4-->
```

- 1.Insert node at end
- 2.Insert node at beg
- 3.Insert node at specific pos
- 4.Delete node at end
- 5.Delete node at beg
- 6.Delete node at specific pos
- 7.Reverse List
- 8.Length of List
- 9.Exit

Enter your choice: 8

Length of List is: 4

Question 28:

Write a program to perform following operations on singly circular linked list:- Create, Traverse, Insertion and Deletion(beg, end , specific), Length of list, Reverse list.

Code:

```
#include <stdio.h>

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

struct node *tail=NULL,*start;
struct node *newNode,*temp;

void allocate_memory(){
    newNode=(struct node*)malloc(sizeof(struct node));
    printf("\nEnter data of node: ");
    scanf("%d",&newNode->data);
    newNode->next=NULL;
}

void node_insert_end(){
    allocate_memory();
    if(tail==NULL){
        tail=newNode;
        newNode->next=tail;
    }
    else{
        temp=tail->next;
        while(temp->next!=tail->next){
            temp=temp->next;
        }
    }
}
```

```

    newNode->next=tail->next;
    temp->next=newNode;
    tail=newNode;
}
}
void node_insert_beg(){
    allocate_memory();
    if(tail==NULL){
        tail=newNode;
        newNode->next=tail;
    }
    else{
        newNode->next=tail->next;
        tail->next=newNode;
    }
}
void node_insert_specific(){
    int i=1,pos;
    allocate_memory();
    if(tail==NULL){
        tail=newNode;
        newNode->next=tail;
    }
    else{
        printf("Enter Position where you want to insert node: ");
        scanf("%d",&pos);
        if(pos==1){
            newNode->next=tail->next;
            tail->next=newNode;

```

```

    }
    else{
        temp=tail->next;
        while(i<pos-1){
            temp=temp->next;
            i++;
        }
        newNode->next=temp->next;
        temp->next=newNode;
    }
}

void node_delete_end(){
    struct node *prev;
    temp=tail->next;
    while(temp->next!=tail->next){
        prev=temp;
        temp=temp->next;
    }
    if(temp==tail->next){
        free(temp);
        tail=NULL;
    }
    else{
        prev->next=temp->next;
        tail=prev;
        free(temp);
    }
}

```

```

void node_delete_beg(){
    if(tail==tail->next){
        free(tail);
        tail=NULL;
    }
    else{
        temp=tail->next;
        tail->next=temp->next;
        free(temp);
    }
}

void node_delete_specific(){
    int i=1,pos;
    struct node *var;
    temp=tail->next;
    printf("Enter Position of node to delete: ");
    scanf("%d",&pos);
    if(pos==1){
        tail->next=temp->next;
        free(temp);
    }
    else{
        while(i<pos-1){
            temp=temp->next;
            i++;
        }
        var=temp->next;
        temp->next=var->next;
        free(var);
    }
}

```

```

    }
}

void print_list(){
    struct node *temp2=tail->next;
    if(tail==NULL)
        printf("\nList is empty...");
    else{
        printf("\nData of Linked List: ");
        while(temp2->next!=tail->next){
            printf("%d-->",temp2->data);
            temp2=temp2->next;
        }
        printf("%d-->",temp2->data);
    }
}

void length(){
    temp=tail->next;
    int count=0;
    while(temp!=tail){
        count++;
        temp=temp->next;
    }
    printf("Length of List is: %d",count+1);
}

int main(void) {
    int choice;
    while(1){
        printf("\n\n1.Insert New Node at end");
        printf("\n2.Insert New Node at begining");
    }
}

```

```
printf("\n3.Insert New Node at specific pos");
printf("\n4.Delete Node from end");
printf("\n5.Delete Node from begining");
printf("\n6.Delete Node from specific pos");
printf("\n7.Length of List");
printf("\n8.Exit");
printf("\nEnter your choice: ");
scanf("%d",&choice);
```

```
if(choice==1){
    node_insert_end();
    printf("\nNode Inserted Successfully :");
    print_list();
}
```

```
if(choice==2){
    node_insert_beg();
    printf("\nNode Inserted Successfully :");
    print_list();
}
```

```
if(choice==3){
    node_insert_specific();
    printf("\nNode Inserted Successfully :");
    print_list();
}
```

```
if(choice==4){
    node_delete_end();
    printf("\nNode Deleted Successfully :");
    print_list();
}
```



```
if(choice==5){
    node_delete_beg();
    printf("\nNode Deleted Successfully :");
    print_list();
}
if(choice==6){
    node_delete_specific();
    printf("\nNode Deleted Successfully :");
    print_list();
}
if(choice==7)
    length();

if(choice==8){
    printf("Program Terminated...");
    break;
}
}
return 0;
}
```

Output:

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Exit
Enter your choice: 1

Enter data of node: 1

Node Inserted Successfully :)
Data of Linked List: 1-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Exit
Enter your choice: 1

Enter data of node: 2

Node Inserted Successfully :)
Data of Linked List: 1-->2-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Exit
Enter your choice: 2
```

```
Enter data of node: 3
```

```
Node Inserted Successfully :)
Data of Linked List: 3-->1-->2-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Exit
Enter your choice: 3
```

```
Enter data of node: 4
```

```
Enter Position where you want to insert node: 2
```

```
Node Inserted Successfully :)
Data of Linked List: 3-->4-->1-->2-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Exit
Enter your choice: 6
Enter Position of node to delete: 3
```

```
Node Deleted Successfully :)
Data of Linked List: 3-->4-->2-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Exit
Enter your choice: 4
```

```
Node Deleted Successfully :)
Data of Linked List: 3-->4-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Exit
Enter your choice: 5
```

```
Node Deleted Successfully :)
Data of Linked List: 4-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Exit
Enter your choice: 7
Length of List is: 1
```

Question 29:

Write a program to perform following operations on doubly circular linked list:- Create, Traverse, Insertion and Deletion(beg, end , specific), Length of list, Reverse list.

Code:

```
#include <stdio.h>

struct node{
    int data;
    struct node *prev;
    struct node *next;
};

struct node *tail=NULL,*start;
struct node *newNode,*temp;

void allocate_memory(){
    newNode=(struct node*)malloc(sizeof(struct node));
    printf("\nEnter data of node: ");
    scanf("%d",&newNode->data);
    newNode->next=NULL;
    newNode->prev=NULL;
}

void node_insert_end(){
    allocate_memory();
    if(tail==NULL){
        tail=newNode;
        newNode->next=tail;
        newNode->prev=tail;
    }
    else{
        newNode->next=tail->next;
        newNode->prev=tail;
```

```

    tail->next=newNode;
    tail=newNode;
}
}

void node_insert_beg(){
    allocate_memory();
    if(tail==NULL){
        tail=newNode;
        newNode->next=tail;
        newNode->prev=tail;
    }
    else{
        newNode->next=tail->next;
        newNode->prev=tail;
        tail->next->prev=newNode;
        tail->next=newNode;
    }
}

void node_insert_specific(){
    int i=1,pos;
    allocate_memory();
    if(tail==NULL){
        tail=newNode;
        newNode->next=tail;
        newNode->prev=tail;
    }
    else{
        printf("Enter Position where you want to insert node: ");
        scanf("%d",&pos);
        temp=tail->next;
        while(i<pos-1){

```

```

        temp=temp->next;
        i++;
    }
    newNode->next=temp->next;
    temp->next->prev=newNode;
    temp->next=newNode;
    newNode->prev=temp;
}
}

void node_delete_end(){
    struct node *temp2=tail;
    if(tail==tail->next)
        tail=NULL;
    else{
        temp=tail->next;
        temp->prev=tail->prev;
        tail->prev->next=temp;
        tail=tail->prev;
    }
    free(temp2);
}

void node_delete_beg(){
    if(tail==tail->next){
        free(tail);
        tail=NULL;
    }
    else{
        temp=tail->next;
        tail->next=temp->next;
        temp->next->prev=tail;
        free(temp);
    }
}

```

```

    }
}
void node_delete_specific(){
    int i=1,pos;
    temp=tail->next;
    struct node *var;
    printf("Enter Position of node to delete: ");
    scanf("%d",&pos);
    while(i<pos-1){
        temp=temp->next;
        i++;
    }
    var=temp->next;
    temp->next=var->next;
    var->next->prev=temp;
    free(var);
}
void print_list(){
    struct node *temp2=tail->next;
    if(tail==NULL)
        printf("\nList is empty...");
    else{
        printf("\nData of Linked List: ");
        while(temp2->next!=tail->next){
            printf("<--%d-->",temp2->data);
            temp2=temp2->next;
        }
        printf("<--%d-->",temp2->data);
    }
}
void length(){

```



```

if(tail==NULL)

    printf("List is Empty..");
else{
temp=tail->next;
int count=0;
while(temp!=tail){

    count++;

    temp=temp->next;
}
printf("Length of List is: %d",count+1);
}
}

int main(void) {
int choice;

while(1){

printf("\n\n1.Insert New Node at end");
printf("\n2.Insert New Node at begining");
printf("\n3.Insert New Node at specific pos");
printf("\n4.Delete Node from end");
printf("\n5.Delete Node from begining");
printf("\n6.Delete Node from specific pos");
printf("\n7.Length of List");
printf("\n8.Exit");
printf("\nEnter your choice: ");
scanf("%d",&choice);

if(choice==1){

    node_insert_end();

    printf("\nNode Inserted Successfully :)");

    print_list();
}

if(choice==2){

```

```

    node_insert_beg();
    printf("\nNode Inserted Successfully :");
    print_list();
}
if(choice==3){
    node_insert_specific();
    printf("\nNode Inserted Successfully :");
    print_list();
}
if(choice==4){
    node_delete_end();
    printf("\nNode Deleted Successfully :");
    print_list();
}
if(choice==5){
    node_delete_beg();
    printf("\nNode Deleted Successfully :");
    print_list();
}
if(choice==6){
    node_delete_specific();
    printf("\nNode Deleted Successfully :");
    print_list();
}
if(choice==7)
    length();
if(choice==8){
    printf("Program Terminated...");
    break;
}
}

```

```
return 0;  
}
```

Output:

```
Data of Linked List: <--1-->  
  
1.Insert New Node at end  
2.Insert New Node at begining  
3.Insert New Node at specific pos  
4.Delete Node from end  
5.Delete Node from begining  
6.Delete Node from specific pos  
7.Length of List  
8.Exit  
Enter your choice: 1  
  
Enter data of node: 2  
  
Node Inserted Successfully :)  
Data of Linked List: <--1--><--2-->
```

```
1.Insert New Node at end  
2.Insert New Node at begining  
3.Insert New Node at specific pos  
4.Delete Node from end  
5.Delete Node from begining  
6.Delete Node from specific pos  
7.Length of List  
8.Exit  
Enter your choice: 2  
  
Enter data of node: 3  
  
Node Inserted Successfully :)  
Data of Linked List: <--3--><--1--><--2-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Exit
Enter your choice: 3

Enter data of node: 4
Enter Position where you want to insert node: 2

Node Inserted Successfully :)
Data of Linked List: <--3--><--4--><--1--><--2-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Exit
Enter your choice: 4

Node Deleted Successfully :)
Data of Linked List: <--3--><--4--><--1-->
```

```
1.Insert New Node at end
2.Insert New Node at begining
3.Insert New Node at specific pos
4.Delete Node from end
5.Delete Node from begining
6.Delete Node from specific pos
7.Length of List
8.Exit
Enter your choice: 5

Node Deleted Successfully :)
Data of Linked List: <--4--><--1-->
```

Data of Linked List: <--4--><--1--><--2--><--5-->



- 1.Insert New Node at end
- 2.Insert New Node at begining
- 3.Insert New Node at specific pos
- 4.Delete Node from end
- 5.Delete Node from begining
- 6.Delete Node from specific pos
- 7.Length of List
- 8.Exit

Enter your choice: 6

Enter Position of node to delete: 2

Node Deleted Successfully :)

Data of Linked List: <--4--><--2--><--5-->

- 1.Insert New Node at end
- 2.Insert New Node at begining
- 3.Insert New Node at specific pos
- 4.Delete Node from end
- 5.Delete Node from begining
- 6.Delete Node from specific pos
- 7.Length of List
- 8.Exit

Enter your choice: 7

Length of List is: 3

Question 30:

Write a program to print middle element of the linked list.

Code:

```
#include <stdio.h>

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

int main(void) {
    int mid1,mid2,length=0,choice=1,i=1;
    struct node *newNode,*temp,*temp2,*start=NULL;
    while(choice){
        newNode=(struct node*)malloc(sizeof(struct node));
        printf("Enter data of node: ");
        scanf("%d",&newNode->data);
        newNode->next=NULL;

        if(start==NULL){
            start=newNode;
            temp=newNode;
        }
        else{
            temp->next=newNode;
            temp=newNode;
        }

        printf("\nDo you want to insert more nodes(1=Yes,0=No): ");
        scanf("%d",&choice);
    }
    temp2=start;
    printf("\nList is: ");
```

```

while(temp2!=NULL){
    printf("%d-->",temp2->data);
    temp2=temp2->next;
    length++;
}

temp2=start;
mid1=length/2;
mid2=mid1+1;
if(length%2==0){
    while(i<mid1){
        temp2=temp2->next;
        i++;
    }
    printf("\nAs length of list is %d(Even), so there will be 2 middle elements...",length);
    printf("\nFirst middle element: %d",temp2->data);
    printf("\nSecond middle element: %d",temp2->next->data);
}
else{
    while(i<mid2){
        temp2=temp2->next;
        i++;
    }
    printf("\nAs length of list is %d(Odd), so there will be 1 middle element...",length);
    printf("\nMiddle element: %d",temp2->data);
}
return 0;
}

```


Output:

```
Enter data of node: 1
Do you want to insert more nodes(1=Yes,0=No): 1
Enter data of node: 2
Do you want to insert more nodes(1=Yes,0=No): 1
Enter data of node: 3
Do you want to insert more nodes(1=Yes,0=No): 1
Enter data of node: 4
Do you want to insert more nodes(1=Yes,0=No): 1
Enter data of node: 5
Do you want to insert more nodes(1=Yes,0=No): 1
Enter data of node: 6
Do you want to insert more nodes(1=Yes,0=No): 0

List is: 1-->2-->3-->4-->5-->6-->
As length of list is 6(Even), so there will be 2 middle elements...
First middle element: 3
Second middle element: 4
```


Question 31:

Write a program to check if a linked list is palindrome or not.

Code:

```
#include <stdio.h>

struct node{
    char data;
    struct node *next;
    struct node *prev;
};

struct node *temp,*temp2,*newNode,*start=NULL;

void display_list(){
    temp2=start;
    while(temp2!=NULL){
        printf("<--%c-->",temp2->data);
        temp2=temp2->next;
    }
}

int main(void) {
    int choice=1,flag=1;
    while(choice){
        newNode=(struct node*)malloc(sizeof(struct node));
        printf("\nEnter data: ");
        scanf("%c",&newNode->data);
        getchar();
        newNode->next=NULL;
        newNode->prev=NULL;
        if(start==NULL)
            start=newNode;
        else{
            newNode->prev=temp;
```

```

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

printf("Do want to enter more nodes(1/0): ");
scanf("%d",&choice);
getchar();
}
display_list();

temp2=start;
while(temp!=NULL && temp2!=NULL && temp!=temp2){
    if(temp2->data==temp->data){
        temp2=temp2->next;
        temp=temp->prev;
        flag=0;
    }
    else{
        flag=1;
        break;
    }
}
if(flag==0)
    printf("\nLinked List is Palindrome :");
else
    printf("\nLinked List is Not Palindrome :(");
return 0;
}

```

Output:

```
Enter data: a
Do want to enter more nodes(1/0): 1

Enter data: b
Do want to enter more nodes(1/0): 1

Enter data: c
Do want to enter more nodes(1/0): 0
<--a--><--b--><--c-->
Linked List is Not Palindrome :( 🐛
```

```
Enter data: a
Do want to enter more nodes(1/0): 1

Enter data: r
Do want to enter more nodes(1/0): 1

Enter data: o
Do want to enter more nodes(1/0): 1

Enter data: r
Do want to enter more nodes(1/0): 1

Enter data: a
Do want to enter more nodes(1/0): 0
<--a--><--r--><--o--><--r--><--a-->
Linked List is Palindrome :) 🐛
```

Question 32:

Write a program to add and multiply two polynomials using linked list.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{ int coef;
  int exp;
  struct node *link; };

void insert_term(struct node **,int,int);
void traverse(struct node *);
void poly_add(struct node **,struct node **,struct node **);
void poly_pdt(struct node **,struct node **,struct node **);

void main()
{
  clrscr();
  struct node *start_p=NULL,*start_q=NULL,*start_r=NULL;
  int i,n,c,e;
  printf("Enter first polynomial!\n");
  printf("Enter no of terms:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  { printf("Enter coefficient: ");
    scanf("%d",&c);
    printf("Enter exponent: ");
    scanf("%d",&e);
    insert_term(&start_p,c,e);
  }

  printf("Enter second polynomial!\n");
  printf("Enter no of terms:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  { printf("Enter coefficient: ");
    scanf("%d",&c);
    printf("Enter exponent: ");
    scanf("%d",&e);
    insert_term(&start_q,c,e);
  }
  printf("First polynomial: ");
  traverse(start_p);
  printf("Second polynomial: ");
  traverse(start_q);
  poly_add(&start_p,&start_q,&start_r);
  printf("Sum of two polynomial:\n");
```

```

traverse(start_r);
start_r=NULL;
poly_pdt(&start_p,&start_q,&start_r);
printf("Product of the two polynomial:\n");
traverse(start_r);
getch();
}

void insert_term(struct node **start,int c,int e)
{ struct node *temp,*temp1,*prev;
  if (*start==NULL)
  {
    temp=(struct node*)malloc(sizeof(struct node));
    if (temp==NULL)
    printf("Node is not created, Term cannot be inserted\n");
    else
    { temp->coef=c;
      temp->exp=e;
      temp->link=NULL;
      *start=temp;
    }
  }
  else
  { temp1=*start;
    while (temp1!=NULL && temp1->exp>e)
    {
      prev=temp1;
      temp1=temp1->link;
    }
    if(temp1==NULL)
    {
      temp=(struct node *)malloc(sizeof(struct node));
      if (temp==NULL)
      printf("Node is not created\n");
      else
      { temp->coef=c;
        temp->exp=e;
        temp->link=NULL;
        prev->link=temp;
      }
    }
    else
    { if(temp1->exp==e)
      temp1->coef=temp1->coef+c;
      else
      { if(temp1==*start)
        { temp=(struct node *)malloc (sizeof (struct node));
          if(temp==NULL)
          printf("Node is not created\n");
          else
          { temp->coef=c;

```

```

    temp->exp=e;
    temp->link=*start;
    *start=temp;
} }
else
{ temp=(struct node *)malloc(sizeof(struct node));
  if (temp==NULL)
    printf("Node is not created\n");
  else
  { temp->coef=c;
    temp->exp=e;
    temp->link=temp1;
    prev->link=temp;
  } }
} }
} }

```

```

void traverse(struct node *start)
{ struct node *temp;
  temp=start;
  if (temp==NULL)
    printf("Empty polynomial\n");
  else
  { while(temp!=NULL)
    { printf("%d x^%d",temp->coef,temp->exp);
      temp=temp->link;
      if(temp!=NULL)
        printf(" + ");
      else
        printf("\n");
    } }
}

```

```

void poly_add(struct node** start_p,struct node **start_q,struct node** start_r)
{ int c,e;
  struct node *pptr,*qptr;
  *start_r=NULL;
  pptr=*start_p;
  qptr=*start_q;
  while(pptr!=NULL && qptr!=NULL)
  { if (pptr->exp==qptr->exp)
    {
      c=pptr->coef+qptr->coef;
      e=pptr->exp;
      insert_term(start_r,c,e);
      pptr=pptr->link;
      qptr=qptr->link;
    }
    else
    { if (pptr->exp>qptr->exp)
      { c=pptr->coef;
        e=pptr->exp;

```

```

        insert_term(start_r,c,e);
        pptr=pptr->link;
    }
else
    { c=qptr->coef;
      e=qptr->exp;
      insert_term(start_r,c,e);
      qptr=qptr->link;
    } }
while(pptr!=NULL)
{
    c=pptr->coef;
    e=pptr->exp;
    insert_term(start_r,c,e);
    pptr=pptr->link;
}
while (qptr!=NULL)
{
    c=qptr->coef;
    e=qptr->exp;
    insert_term(start_r,c,e);
    qptr=qptr->link;
}}

void poly_pdt(struct node ** start_p,struct node **start_q,struct node** start_r)
{
    int c,e;
    struct node *pptr,*qptr;
    *start_r=NULL;
    pptr=*start_p;
    qptr=*start_q;
    if (*start_p==NULL && *start_q==NULL)
        printf("\nMultiplication of polynomial is not possible!\n");
    else
    { while(pptr!=NULL)
      {
          qptr=*start_q;
          while(qptr!=NULL)
          {
              c=pptr->coef*qptr->coef;
              e=pptr->exp+qptr->exp;
              insert_term(start_r,c,e);
              qptr=qptr->link;
          }
          pptr=pptr->link;
      }
    }
}

```

OUTPUT:

```
> make -s
> ./main
Enter first polynomial!
Enter no of terms:3
Enter coefficient: 3
Enter exponent: 2
Enter coefficient: 2
Enter exponent: 1
Enter coefficient: 5
Enter exponent: 0
Enter second polynomial!
Enter no of terms:3
Enter coefficient: 4
Enter exponent: 3
Enter coefficient: 4
Enter exponent: 2
Enter coefficient: 7
Enter exponent: 1
First polynomial: 3 x^2 + 2 x^1 + 5 x^0
Second polynomial: 4 x^3 + 4 x^2 + 7 x^1
Sum of two polynomial:
4 x^3 + 7 x^2 + 9 x^1 + 5 x^0
Product of the two polynomial:
12 x^5 + 20 x^4 + 49 x^3 + 34 x^2 + 35 x^1
> 
```


Question 33:

Write a program to sort a linked list and add a node in that sorted linked list.

Code:

```
#include <stdio.h>

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

struct node *newNode,*temp,*start=NULL;

void print_list(){
    struct node *temp2;
    printf("\nList is: ");
    temp2=start;
    while(temp2!=NULL){
        printf("%d-->",temp2->data);
        temp2=temp2->next;
    }
}

int main(void) {
    int choice=1;
    while(choice){
        newNode=(struct node*)malloc(sizeof(struct node));
        printf("Enter data of node: ");
        scanf("%d",&newNode->data);
        newNode->next=NULL;
```

```

if(start==NULL){
    start=newNode;
    temp=newNode;
}
else{
    temp->next=newNode;
    temp=newNode;
}
printf("\nDo you want to enter more nodes(1=Yes,0=No): ");
scanf("%d",&choice);
}
print_list();

struct node *i=start,*j;
while(i!=NULL){
    j=i->next;
    while(j!=NULL){
        if(i->data > j->data){
            int c=i->data;
            i->data=j->data;
            j->data=c;
        }
        j=j->next;
    }
    i=i->next;
}
printf("\n\n---List After sorting---");

```

```

print_list();

struct node *prev;
i=start;
int flag=1;
newNode=(struct node*)malloc(sizeof(struct node));
printf("\n\nEnter data of node to insert: ");
scanf("%d",&newNode->data);
newNode->next=NULL;

while(newNode->data > i->data){
    prev=i;
    i=i->next;
    flag=0;
}
if(flag==1){
    newNode->next=i;
    start=newNode;
}
else{
    prev->next=newNode;
    newNode->next=i;
}

printf("\nNode Inserted Successfully :)");
print_list();
return 0;
}

```

Output:

```
➤ ./main
Enter data of node: 8

Do you want to enter more nodes(1=Yes,0=No): 1
Enter data of node: 5

Do you want to enter more nodes(1=Yes,0=No): 1
Enter data of node: 4

Do you want to enter more nodes(1=Yes,0=No): 1
Enter data of node: 1

Do you want to enter more nodes(1=Yes,0=No): 0

List is: 8-->5-->4-->1-->

---List After sorting---
List is: 1-->4-->5-->8-->

Enter data of node to insert: 6

Node Inserted Successfully :)
List is: 1-->4-->5-->6-->8-->➤
```

Question 34:

Write a program to implement Stack as an Array.

Code:

```
#include <stdio.h>

int stack[5];

int top=-1;

int n= sizeof(stack)/sizeof(int);

void push(int el){
    if(top==n-1)
        printf("\nStack Overflow!!!");
    else{
        ++top;
        stack[top]=el;
    }
}

void pop(){
    if(top== -1)
        printf("\nStack Underflow!!!");
    else{
        int temp=stack[top];
        top--;
        printf("\nPopped item is: %d",temp);
    }
}

void display_stack(){
    printf("\nStack is: ");
    for(int i=top;i>=0;i--)
        printf("%d| ",stack[i]);
}

int main(void)
```

```

{
int choice;
while(1){
    printf("\n\n1.Insert Item in Stack");
    printf("\n\n2.Delete Item from Stack");
    printf("\n\n3.Print Items of Stack");
    printf("\n\n4.Exit");
    printf("\n\nEnter your choice: ");
    scanf("%d",&choice);

    if(choice==1){
        int el;
        printf("\n\nEnter item: ");
        scanf("%d",&el);
        push(el);
        display_stack();
    }
    if(choice==2){
        pop();
        display_stack();
    }
    if(choice==3){
        display_stack();
    }
    if(choice==4){
        printf("Program Terminated...");
        break;
    }
}
return 0;
}

```

Output:

```
1.Insert Item in Stack
2.Delete Item from Stack
3.Print Items of Stack
4.Exit
Enter your choice: 1
```

```
Enter item: 1
```

```
Stack is: 1|
```

```
1.Insert Item in Stack
2.Delete Item from Stack
3.Print Items of Stack
4.Exit
Enter your choice: 1
```

```
Enter item: 2
```

```
Stack is: 2| 1|
```

```
1.Insert Item in Stack
2.Delete Item from Stack
3.Print Items of Stack
4.Exit
Enter your choice: 1
```

```
Enter item: 3
```

```
Stack is: 3| 2| 1|
```

```
1.Insert Item in Stack
2.Delete Item from Stack
3.Print Items of Stack
4.Exit
Enter your choice: 2
```

```
Popped item is: 3
Stack is: 2| 1|
```

Question 35:

Write a program to implement Stack as a Linked List.

Code:

```
#include <stdio.h>

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

struct node *top=NULL,*temp,*newNode;

void push(int el){
    newNode=(struct node*)malloc(sizeof(struct node));
    newNode->data=el;
    newNode->next=NULL;
    if(top==NULL)
        top=newNode;
    else{
        newNode->next=top;
        top=newNode;
    }
}

void pop(){
    if(top==NULL)
        printf("\nCan't delete as");
    else{
        temp=top;
        top=top->next;
        printf("\nPopped item is: %d",temp->data);
    }
}
```



```

    free(temp);
}
}
void display_stack(){
    temp=top;
    if(top==NULL)
        printf("\nStack is empty...");
    else{
        printf("\nStack is: ");
        while(temp!=NULL){
            printf("%d| ",temp->data);
            temp=temp->next;
        }
    }
}
int main(void) {
    int choice;
    while(1){
        printf("\n\n1.Insert Item in Stack");
        printf("\n2.Delete Item from Stack");
        printf("\n3.Print Items of Stack");
        printf("\n4.Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);

        if(choice==1){
            int el;
            printf("\nEnter item: ");

```

```
scanf("%d",&el);
push(el);
display_stack();
}
if(choice==2){
    pop();
    display_stack();
}
if(choice==3){
    display_stack();
}
if(choice==4){
    printf("Program Terminated...");
    break;
}
}
return 0;
}
```

Output:

```
1.Insert Item in Stack
2.Delete Item from Stack
3.Print Items of Stack
4.Exit
Enter your choice: 1
```

```
Enter item: 1
```

```
Stack is: 1|
```

```
1.Insert Item in Stack
2.Delete Item from Stack
3.Print Items of Stack
4.Exit
Enter your choice: 1
```

```
Enter item: 2
```

```
Stack is: 2| 1|
```

```
1.Insert Item in Stack
2.Delete Item from Stack
3.Print Items of Stack
4.Exit
Enter your choice: 1
```

```
Enter item: 3
```

```
Stack is: 3| 2| 1|
```

```
1.Insert Item in Stack
2.Delete Item from Stack
3.Print Items of Stack
4.Exit
Enter your choice: 2
```

```
Popped item is: 3
Stack is: 2| 1|
```

Question 36:

Write a program to Reverse a String using Stack.

```
#include <stdio.h>

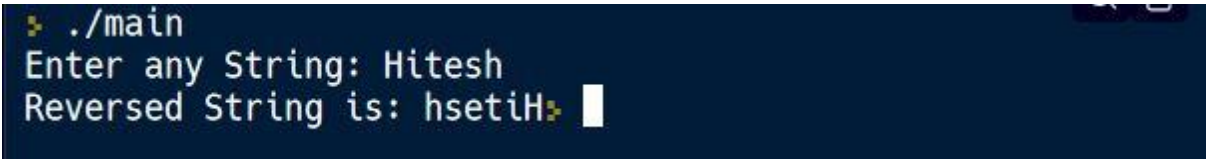
char stack[10],output[10],top=-1;

void push(char el){
    top++;
    stack[top]=el;
}

char pop(){
    char item=stack[top];
    top--;
    return item;
}

int main(void) {
    char str[6];
    printf("Enter any String: ");
    scanf("%s",str);
    int len=sizeof(str)/sizeof(char);
    for(int i=0;i<len;i++)
        push(str[i]);
    printf("Reversed String is: ");
    for(int i=top;i>=0;i--){
        char el=pop();
        printf("%c",el);
    }
}
```

Output:



```
./main
Enter any String: Hitesh
Reversed String is: hsetiH
```

Question 37

Write a program to Convert Infix expression to Postfix.

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

char stack[100];
int top = -1;
void push(char x)
{
    stack[++top] = x;
}
char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}
int priority(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/')
        return 2;
    return 0;
}

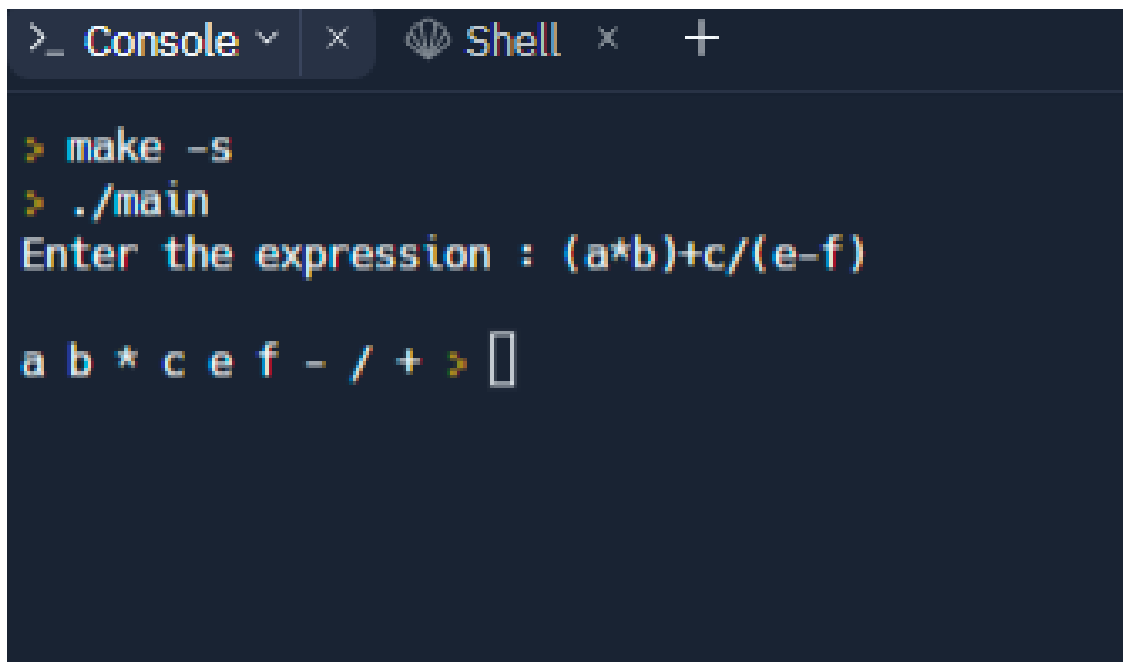
int main()
{
    char exp[100];
    char *e, x;
    printf("Enter the expression : ");
    scanf("%s",exp);
    printf("\n");
    e = exp;
    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c ",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
```

```

    { while((x = pop()) != '(')
        printf("%c ", x);    }
    else
    {   while(priority(stack[top]) >= priority(*e))
        printf("%c ",pop());
        push(*e);    }
    e++;
}
while(top != -1)
{
    printf("%c ",pop());
}return 0;
}

```

OUTPUT:



```

>_ Console x Shell x +
> make -s
> ./main
Enter the expression : (a*b)+c/(e-f)
a b * c e f - / + > 

```

Question 38

Write a program to Convert Infix expression to Prefix.

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

#define MAX 100

int top = -1;
char stack[MAX];

int isFull ()
{
    return top == MAX - 1;
}

int isEmpty ()
{
    return top == -1;
}

void push (char item)
{
    if (isFull ())
        return;
    top++;
    stack[top] = item;
}

int pop ()
{
    if (isEmpty ())
        return INT_MIN;
    return stack[top--];
}

int peek ()
{
    if (isEmpty ())
        return INT_MIN;
    return stack[top];
}
```

```

int checkIfOperand (char ch)
{
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}

int precedence (char ch)
{
    switch (ch)
    {
        case '+':
        case '-':
            return 1;

        case '*':
        case '/':
            return 2;

        case '^':
            return 3;
    }
    return -1;
}

int getPostfix (char *expression){
    int i, j;
    for (i = 0, j = -1; expression[i]; ++i)
    {

        if (checkIfOperand (expression[i]))
            expression[++j] = expression[i];

        else if (expression[i] == '(')
            push (expression[i]);

        else if (expression[i] == ')')
        {
            while (!isEmpty (stack) && peek (stack) != '(')
                expression[++j] = pop (stack);
            if (!isEmpty (stack) && peek (stack) != '(')
                return -1;
            else
                pop (stack);
        }
        else
        {
            while (!isEmpty (stack)
                && precedence (expression[i]) <= precedence (peek (stack)))

```



```

        expression[++j] = pop (stack);
        push (expression[i]);
    }
}
while (!isEmpty (stack))
    expression[++j] = pop (stack);

expression[++j] = '\0'; }

void reverse (char *exp) {
    int size = strlen (exp);
    int j = size, i = 0;
    char temp[size];
    temp[j--] = '\0';
    while (exp[i] != '\0')
    {
        temp[j] = exp[i];
        j--;
        i++;
    }
    strcpy (exp, temp); }

void brackets (char *exp)
{   int i = 0;
    while (exp[i] != '\0')
    {
        if (exp[i] == '(')
            exp[i] = ')';
        else if (exp[i] == ')')
            exp[i] = '(';
        i++;
    } }

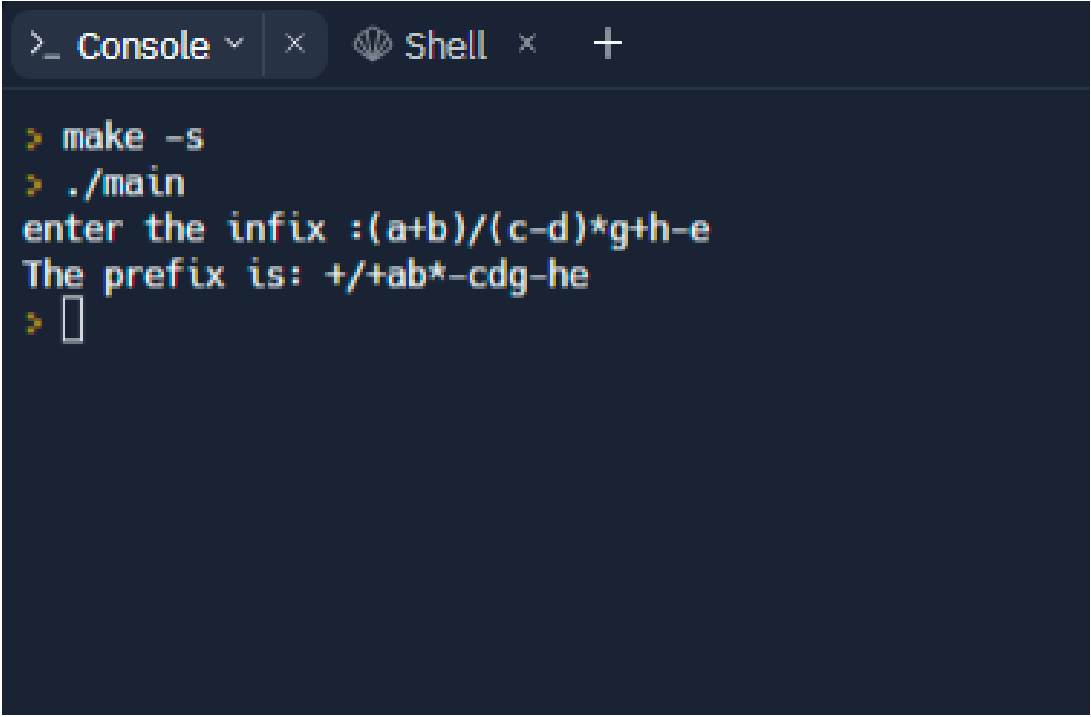
void InfixtoPrefix (char *exp)
{
    int size = strlen (exp);
    reverse (exp);
    brackets (exp);
    getPostfix (exp);
    reverse (exp);
}

int main ()
{
    char expression[20];
    printf ("enter the infix :");
    scanf ("%s", expression);

```

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

OUTPUT:



```
>_ Console x Shell x +  
  
> make -s  
> ./main  
enter the infix :(a+b)/(c-d)*g+h-e  
The prefix is: +/+ab*-cdg-he  
> 
```

Question 39

Write a program to implement Queue using Arrays.

```
#include <stdio.h>

int queue[5];

int n= sizeof(queue)/sizeof(int);

int front=-1;

int rear=-1;

void enqueue(int el){

    if(rear==n-1)

        printf("\nQueue is Full!!");

    else if(front== -1 && rear== -1){

        front=0;

        rear=0;

        queue[rear]=el;

    }

    else{

        rear++;

        queue[rear]=el;

    }

}

void dequeue(){

    if(front==rear){

        printf("\nDequeued element is: %d",queue[front]);

        front=-1;

        rear=-1;

    }

    else{

        printf("\nDequeued element is: %d",queue[front]);

        front++;

    }

}
```

```

}

void peek(){
    if(front==-1 && rear==-1)
        printf("\nQueue is Empty..");
    else
        printf("\nTopmost element is: %d",queue[front]);
}

void print_queue(){
    if(front==-1 && rear==-1)
        printf("\nQueue is Empty..");
    else{
        printf("\nQueue is: ");
        for(int i=front;i<=rear;i++)
            printf("%d--",queue[i]);
    }
}

int main(void) {
    int choice;
    while(1){
        printf("\n\n1.Insert Item in Queue");
        printf("\n2.Delete Item from Queue");
        printf("\n3.Print Topmost element");
        printf("\n4.Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);

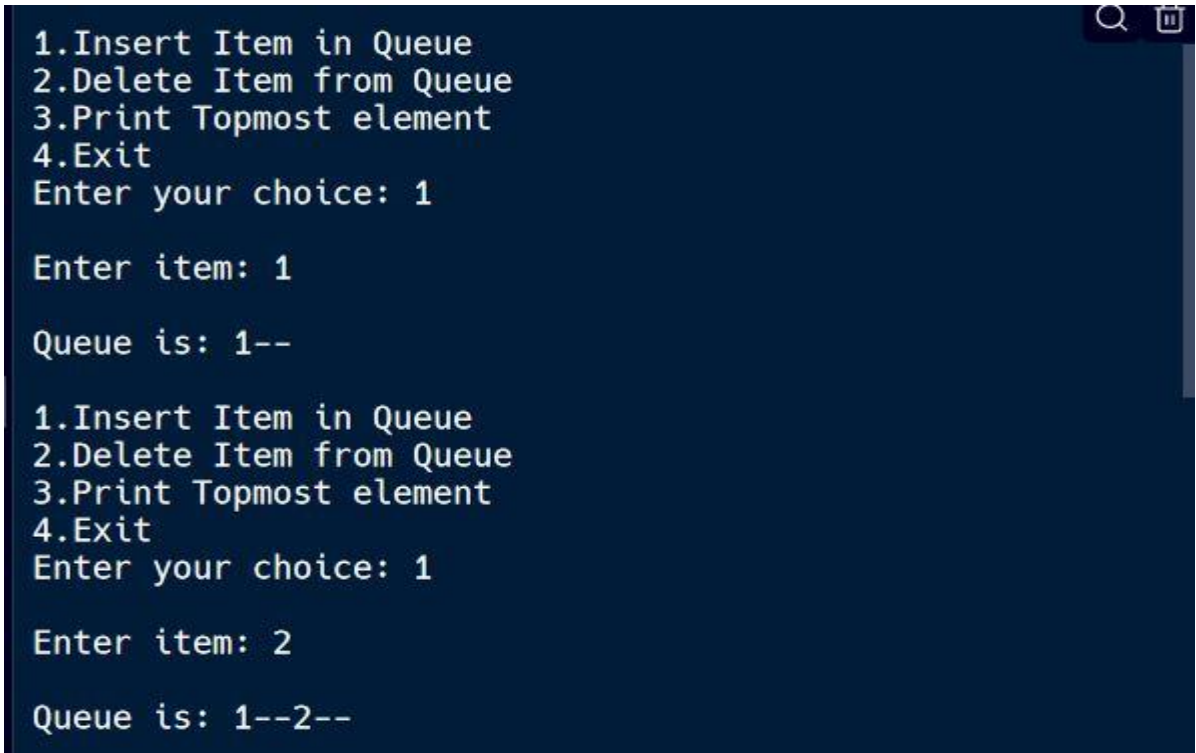
        if(choice==1){
            int el;
            printf("\nEnter item: ");
            scanf("%d",&el);
            enqueue(el);

```

```

    print_queue();
}
if(choice==2){
    dequeue();
    print_queue();
}
if(choice==3){
    peek();
}
if(choice==4){
    printf("Program Terminated...");
    break;
}
}
return 0;
}

```



```

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 1

Enter item: 1

Queue is: 1--

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 1

Enter item: 2

Queue is: 1--2--

```

```
1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
```

Enter your choice: 1

Enter item: 3

Queue is: 1--2--3--

```
1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
```

Enter your choice: 2

Dequeued element is: 1

Queue is: 2--3--

```
1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
```

Enter your choice: 3

Topmost element is: 2

Question 40

Write a program to implement Queue using Linked List.

```
#include <stdio.h>

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

struct node *front=NULL,*rear=NULL,*newNode,*temp;

void enqueue(int el){
    newNode=(struct node*)malloc(sizeof(struct node));
    newNode->data=el;
    newNode->next=NULL;
    if(rear==NULL){
        rear=newNode;
        front=newNode;
    }
    else{
        rear->next=newNode;
        rear=newNode;
    }
}

void dequeue(){
    temp=front;
    printf("\nDequeued element is: %d",front->data);
    front=front->next;
    free(temp);
}

void peek(){
    if(front==NULL && rear==NULL)
        printf("\nQueue is Empty..");
}
```

```

else
    printf("\nTopmost element is: %d",front->data);
}
void print_queue(){
    if(front==NULL && rear==NULL)
        printf("\nQueue is Empty..");
    else{
        temp=front;
        printf("\nQueue is: ");
        while(temp!=NULL){
            printf("%d--",temp->data);
            temp=temp->next;
        }
    }
}
int main(void) {
    int choice;
    while(1){
        printf("\n\n1.Insert Item in Queue");
        printf("\n2.Delete Item from Queue");
        printf("\n3.Print Topmost element");
        printf("\n4.Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);

        if(choice==1){
            int el;
            printf("\nEnter item: ");
            scanf("%d",&el);
            enqueue(el);
            print_queue();

```



```

    }
    if(choice==2){
        dequeue();
        print_queue();
    }
    if(choice==3){
        peek();
    }
    if(choice==4){
        printf("Program Terminated...");
        break;
    }
}
return 0;
}

```

```

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 1

Enter item: 5

Queue is: 5--

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 1

Enter item: 4

Queue is: 5--4--

```

```
1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 1
```

```
Enter item: 7
```

```
Queue is: 5--4--7--
```

```
1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 2
```

```
Dequeued element is: 5
Queue is: 4--7--
```

```
1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 3
```

```
Topmost element is: 4
```

Question 41

Write a program to implement Circular Queue using Arrays.

```
# include<stdio.h>

int queue[5];

int n=sizeof(queue)/sizeof(int);

int front=-1;

int rear=-1;

void enqueue(int el){

    if((rear+1) % n== front)

        printf("\nQueue is Full!!");

    else if(front== -1 && rear== -1){

        front=0;

        rear=0;

        queue[rear]=el;

    }

    else{

        rear=(rear+1) % n;

        queue[rear]=el;

    }

}

void dequeue(){

    if(front==rear){

        printf("\nDequeued element is: %d",queue[front]);

        front=-1;

        rear=-1;

    }

    else{

        printf("\nDequeued element is: %d",queue[front]);

        front= (front+1) % n;

    }

}
```

```

}

void peek(){
    if(front==-1 && rear==-1)
        printf("\nQueue is Empty..");
    else
        printf("\nTopmost element is: %d",queue[front]);
}

void print_queue(){
    if(front==-1 && rear==-1)
        printf("\nQueue is Empty..");
    else{
        printf("\nQueue is: ");
        int i=front;
        while(i!=rear){
            printf("%d--",queue[i]);
            i=(i+1)%n;
        }
        printf("%d--",queue[i]);
    }
}

int main(void) {
    int choice;
    while(1){
        printf("\n\n1.Insert Item in Queue");
        printf("\n2.Delete Item from Queue");
        printf("\n3.Print Topmost element");
        printf("\n4.Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);
        if(choice==1){
            int el;

```

```

    printf("\nEnter item: ");
    scanf("%d",&el);
    enqueue(el);
    print_queue();
}
if(choice==2){
    dequeue();
    print_queue();
}
if(choice==3){
    peek();
}
if(choice==4){
    printf("Program Terminated...");
    break;
}
}
}

```

```

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 1

Enter item: 4

Queue is: 1--2--3--4--

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 1

Enter item: 5

Queue is: 1--2--3--4--5--

```

```
1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 1

Enter item: 7

Queue is Full!!
Queue is: 1--2--3--4--5--

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 2

Dequeued element is: 1
Queue is: 2--3--4--5--
```

```
Queue is: 2--3--4--5--6--

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 3

Topmost element is: 2

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 2

Dequeued element is: 2
Queue is: 3--4--5--6--
```

Question 42

Write a program to implement Circular Queue using Linked List.

```
#include <stdio.h>

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

struct node *front=NULL,*rear=NULL,*newNode,*temp;

void enqueue(int el){
    newNode=(struct node*)malloc(sizeof(struct node));
    newNode->data=el;
    newNode->next=NULL;

    if(rear==NULL){
        rear=newNode;
        front=newNode;
    }
    else{
        rear->next=newNode;
        rear=newNode;
    }
}

void dequeue(){
    temp=front;
    printf("\nDequeued element is: %d",front->data);
    front=front->next;
    free(temp);
}

void peek(){
    if(front==NULL && rear==NULL)
```

```

    printf("\nQueue is Empty..");
else
    printf("\nTopmost element is: %d",front->data);
}
void print_queue(){
    if(front==NULL && rear==NULL)
        printf("\nQueue is Empty..");
    else{
        temp=front;
        printf("\nQueue is: ");
        while(temp!=NULL){
            printf("%d--",temp->data);
            temp=temp->next;
        }
    }
}
int main(void) {
    int choice;
    while(1){
        printf("\n\n1.Insert Item in Queue");
        printf("\n2.Delete Item from Queue");
        printf("\n3.Print Topmost element");
        printf("\n4.Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);

        if(choice==1){
            int el;
            printf("\nEnter item: ");
            scanf("%d",&el);
            enqueue(el);

```



```
    print_queue();  
}  
if(choice==2){  
    dequeue();  
    print_queue();  
}  
if(choice==3){  
    peek();  
}  
if(choice==4){  
    printf("Program Terminated...");  
    break;  
}  
}  
return 0;  
}
```

Queue is: 1--3--

- 1.Insert Item in Queue
- 2.Delete Item from Queue
- 3.Print Topmost element
- 4.Exit

Enter your choice: 1

Enter item: 5

Queue is: 1--3--5--

- 1.Insert Item in Queue
- 2.Delete Item from Queue
- 3.Print Topmost element
- 4.Exit

Enter your choice: 1

Enter item: 8

Queue is: 1--3--5--8--

- 1.Insert Item in Queue
- 2.Delete Item from Queue
- 3.Print Topmost element
- 4.Exit

Enter your choice: 2

Dequeued element is: 1

Queue is: 3--5--8--

- 1.Insert Item in Queue
- 2.Delete Item from Queue
- 3.Print Topmost element
- 4.Exit

Enter your choice: 3

Topmost element is: 3

Question 43

Write a program to implement Queue using Stack (Multi-Stack).

```
#include<stdio.h>

#define N 5

int stack1[N],stack2[N];
int top1=-1,top2=-1,count=0;

void push1(int num){
    if(top1==N-1){
        printf("Overflow\n");
    }
    else{
        top1++;
        stack1[top1]=num;
    }
}

int pop1(){
    return (stack1[top1--]);
}

int pop2(){
    return (stack2[top2--]);
}

void push2(int num){
    if(top2==N-1){
        printf("Overflow\n");
    }
    else{
        top2++;
        stack2[top2]=num;
    }
}
```

```

void enqueue(int el){
    push1(el);
    count++;
}

void dequeue(){
    if(top1== -1 && top2== -1){
        printf("Queue is empty \n");
    }
    else{
        for(int i=0;i<count;i++){
            push2(pop1());
        }
        printf("Deleted element is: %d \n", pop2());
        count--;
        for(int i=0;i<count;i++){
            push1(pop2());
        }
    }
}

void peek(){
    if(top1== -1){
        printf("\nQueue is empty!!");
    }
    else{
        printf("\nTopmost element is: %d ",stack1[0]);
    }
}

void print_queue(){
    if(top1 == -1){
        printf("No element\n");
    }
}

```

```

else{
    for(int i=0;i<=top1;i++){
        printf("%d--",stack1[i]);
    }
}
}

int main(){
    int choice;

    // printf("\n\n1.Enqueue Operation \n2.Dequeue Operation \n3.Peek Operation \n4.Display
Queue\n5.Exit\n");

    while(1){
        printf("\n\n1.Insert Item in Queue");
        printf("\n\n2.Delete Item from Queue");
        printf("\n\n3.Print Topmost element");
        printf("\n\n4.Exit");
        printf("\n\nEnter your choice: ");
        scanf("%d",&choice);

        if(choice==1){
            int el;
            printf("\n\nEnter item: ");
            scanf("%d",&el);
            enqueue(el);
            print_queue();
        }
        if(choice==2){
            dequeue();
            print_queue();
        }
        if(choice==3){

```

```

    peek();
}
if(choice==4){
    printf("Program Terminated...");
    break;
}
}
return 0;
}

```

```

Enter your choice: 1
Enter item: 2
1--2--

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 1
Enter item: 3
1--2--3--

```

```

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 2
Deleted element is: 1
2--3--

1.Insert Item in Queue
2.Delete Item from Queue
3.Print Topmost element
4.Exit
Enter your choice: 3
Topmost element is: 2

```

Question 45

**Write a program to create a Binary Tree along with Traversal
(INORDER,PRE-ORDER,POST-ORDER)**

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *left,*right;
};
struct node *create(){
    struct node *newnode;
    int x;
    newnode=(struct node *) malloc (sizeof(struct node));
    printf("Enter data (Press -1 for no node): ");
    scanf("%d",&x);
    if(x==-1){
        return NULL;
    }
    newnode->data=x;
    printf("Enter the left child: \n");
    newnode->left=create();
    printf("Enter the right child: \n");
    newnode->right=create();
    return newnode;
}
void preorder(struct node *root){
    if(root==NULL){
        return;
    }
}
```

```

    printf("%d ",root->data);
    preorder(root->left);
    preorder(root->right);
}

void inorder(struct node *root){
    if(root==NULL){
        return;
    }
    inorder(root->left);
    printf("%d ",root->data);
    inorder(root->right);
}

void postorder(struct node *root){
    if(root==NULL){
        return;
    }
    postorder(root->left);
    postorder(root->right);
    printf("%d ",root->data);
}

int main(){
    struct node *root;
    root=create();
    printf("\nPreorder is: ");
    preorder(root);
    printf("\nInorder is: ");
    inorder(root);
    printf("\nPostorder is: ");
    postorder(root);
    return 0;
}

```



```
> make -s
> ./main
Enter data (Press -1 for no node): 1
Enter the left child:
Enter data (Press -1 for no node): 2
Enter the left child:
Enter data (Press -1 for no node): -1
Enter the right child:
Enter data (Press -1 for no node): -1
Enter the right child:
Enter data (Press -1 for no node): 3
Enter the left child:
Enter data (Press -1 for no node): -1
Enter the right child:
Enter data (Press -1 for no node): -1

Preorder is: 1 2 3
Inorder is: 2 1 3
Postorder is: 2 3 1 > 
```

Question 46

WAP to implement following operations in BST:

- 1. Insertion**
- 2. Deletion**
- 3. Traverse-(Inorder,Postorder,Preorder)**

```
#include <stdio.h>

#include <stdlib.h>

struct node {
    int data;
    struct node* left,*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;
}

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

void preorder(struct node* root) {
```

```

    if (root==NULL) {
        return;
    }
    printf("%d ",root->data);
    preorder(root->left);
    preorder(root->right);
}

void inorder(struct node* root) {
    if (root==NULL) {
        return;
    }
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

void postorder(struct node* root) {
    if (root == NULL) {
        return;
    }
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}

struct node* findMinNode(struct node* node) {
    struct node* current = node;
    while (current && current->left != NULL) {
        current=current->left;
    }
    return current;
}

struct node* deleteNode(struct node* root, int data) {

```

```

if (root == NULL) {
    return root;
}
if (data < root->data) {
    root->left = deleteNode(root->left, data);
} else if (data > root->data) {
    root->right = deleteNode(root->right, data);
} else {
    // Node to be deleted found
    // Case 1: No child or only one child
    if (root->left==NULL) {
        struct node* temp=root->right;
        free(root);
        return temp;
    } else if (root->right==NULL) {
        struct node* temp=root->left;
        free(root);
        return temp;
    }
    // Case 2: Two children
    struct node* temp=findMinNode(root->right);
    root->data=temp->data;
    root->right=deleteNode(root->right, temp->data);
}
return root;
}

int main() {
    struct node* root = NULL;
    int n, data;
    printf("Enter the number of nodes in the binary search tree: ");
    scanf("%d", &n);

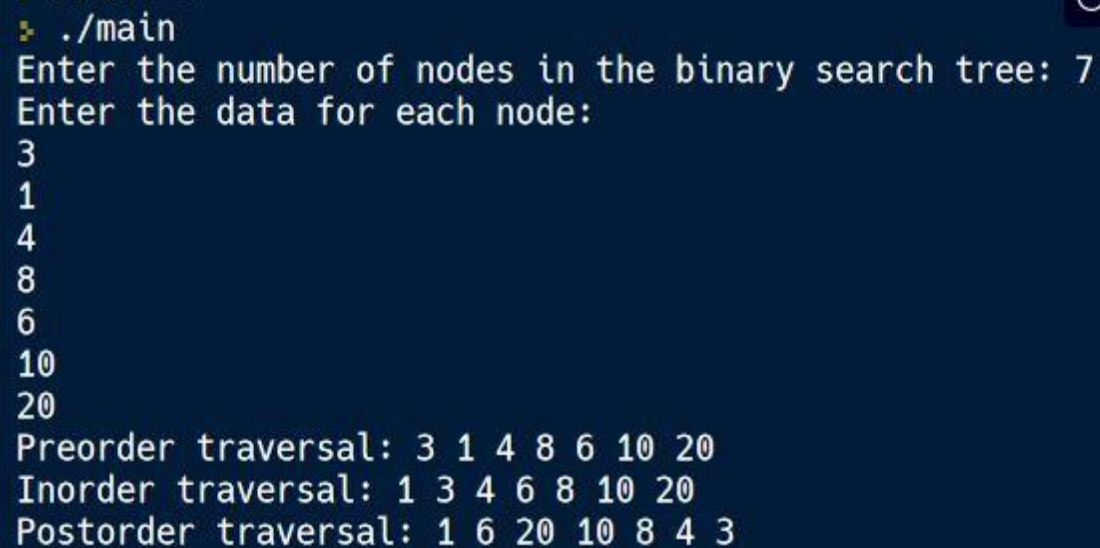
```

```

printf("Enter the data for each node:\n");
for (int i = 0; i < n; i++) {
    scanf("%d",&data);
    root=insert(root, data);
}
printf("Preorder traversal: ");
preorder(root);
printf("\nInorder traversal: ");
inorder(root);
printf("\nPostorder traversal: ");
postorder(root);

int key;
printf("\nEnter the data to delete: ");
scanf("%d",&key);
root=deleteNode(root, key);
printf("\nInorder traversal after deletion: ");
inorder(root);
printf("\n");
return 0;
}

```



```

./main
Enter the number of nodes in the binary search tree: 7
Enter the data for each node:
3
1
4
8
6
10
20
Preorder traversal: 3 1 4 8 6 10 20
Inorder traversal: 1 3 4 6 8 10 20
Postorder traversal: 1 6 20 10 8 4 3

```

Question 47

WAP TO IMPLEMENT FOLLOWING OPERATIONS ON AVL TREES:

- 1. Insertion**
- 2. Deletion**
- 3. Traversal(Inorder,Preorder,Postorder)**

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

```

    newNode->key = key;
    newNode->left = NULL;
    newNode->right = NULL;
    newNode->height = 1;
    return newNode;
}

struct Node* rotateRight(struct Node* y) {
    struct Node* x = y->left;
    struct Node* T2 = x->right;
    x->right = y;
    y->left = T2;
    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;
    return x;
}

struct Node* rotateLeft(struct Node* x) {
    struct Node* y = x->right;
    struct Node* T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;
    return y;
}

struct Node* insert(struct Node* node, int key) {
    if (node == NULL) {
        return createNode(key);
    }
    if (key < node->key) {
        node->left = insert(node->left, key);
    } else if (key > node->key) {

```

```

        node->right = insert(node->right, key);
    } else {
        // Duplicate keys are not allowed in AVL tree
        return node;
    }
    node->height = 1 + max(height(node->left), height(node->right));
    int balanceFactor = getBalanceFactor(node);
    if (balanceFactor > 1) {
        // Left Left case
        if (key < node->left->key) {
            return rotateRight(node);
        }
        // Left Right case
        else if (key > node->left->key) {
            node->left = rotateLeft(node->left);
            return rotateRight(node);
        }
    }
    if (balanceFactor < -1) {
        // Right Right case
        if (key > node->right->key) {
            return rotateLeft(node);
        }
        // Right Left case
        else if (key < node->right->key) {
            node->right = rotateRight(node->right);
            return rotateLeft(node);
        }
    }
    return node;
}

```



```

struct Node* findMinValueNode(struct Node* node) {
    struct Node* current = node;
    while (current->left != NULL) {
        current = current->left;
    }
    return current;
}

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

```

```

        return root;
    }
    root->height=1+max(height(root->left),height(root->right));
    int balanceFactor=getBalanceFactor(root);
    if (balanceFactor>1) {
        if (getBalanceFactor(root->left)>=0) {
            return rotateRight(root);
        } else {
            root->left=rotateLeft(root->left);
            return rotateRight(root);
        }
    }
    if(balanceFactor < -1){
        if(getBalanceFactor(root->right)<=0) {
            return rotateLeft(root);
        }
        else{
            root->right=rotateRight(root->right);
            return rotateLeft(root);
        }
    }
    return root;
}

void preorderTraversal(struct Node* root) {
    if(root==NULL){
        return;
    }
    printf("%d ",root->key);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

```

```

void inorderTraversal(struct Node* root) {
    if(root==NULL){
        return;
    }
    inorderTraversal(root->left);
    printf("%d ",root->key);
    inorderTraversal(root->right);
}

void postorderTraversal(struct Node* root) {
    if(root==NULL){
        return;
    }
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d ",root->key);
}

int main() {
    struct Node* root = NULL;
    int choice,key;
    printf("Enter the elements to be inserted into the AVL tree (-1 to stop):\n");
    while(1){
        printf("Enter element: ");
        scanf("%d",&key);
        if(key==-1) {
            break;
        }
        root=insert(root,key);
    }
    printf("\nPreorder traversal of the AVL tree: ");
    preorderTraversal(root);
    printf("\nInorder traversal of the AVL tree: ");

```

```

inorderTraversal(root);

printf("\nPostorder traversal of the AVL tree: ");

postorderTraversal(root);

printf("\nEnter the element to be deleted from the AVL tree: ");

scanf("%d", &key);

root = deleteNode(root, key);

printf("\nInorder traversal of the AVL tree after deletion: ");

inorderTraversal(root);

printf("\n");

return 0;
}

```

```

./main
Enter the elements to be inserted into the AVL tree (-1 to stop):
Enter element: 2
Enter element: 3
Enter element: 1
Enter element: 7
Enter element: 5
Enter element: 98
Enter element: 13
Enter element: 14
Enter element: -1

Preorder traversal of the AVL tree: 5 2 1 3 13 7 98 14
Inorder traversal of the AVL tree: 1 2 3 5 7 13 14 98
Postorder traversal of the AVL tree: 1 3 2 7 14 98 13 5
Enter the element to be deleted from the AVL tree: 13

Inorder traversal of the AVL tree after deletion: 1 2 3 5 7 14 98

```

Question 48

WAP to store the graph information in form of adjacency matrix.

```
#include <stdio.h>

#include <stdlib.h>

#include<stdbool.h>

int size;

int** graph;

int* visited;

void initializeGraph(int n) {

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

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

        graph[i] = (int*) malloc(n* sizeof(int));

    }

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

        for(int j=0 ;j< n ;j++){

            graph[i][j]= 0;

        }

    }

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

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

        visited[i] = 0;

    }

}

void createNodes(){

    int n;

    printf("Enter the number of nodes you want to add in the graph : ");

    scanf("%d",&n);

    size = n;

    initializeGraph(size);

}
```

```

printf("Choose which nodes are connected!\n For connected nodes , enter 1 else 0 :
\n");
for(int i=0 ;i<n ;i++){
    for(int j =0 ;j<n ;j++){
        int v;
        if(graph[i][j] == 0){
            printf("Is node %d connected to node %d:" , i ,j);
            scanf("%d", &v);
            graph[i][j] = v;
            graph[j][i] = v;
        }
    } }
}

struct queue {
    int size;
    int front;
    int rear;
    int *arr;
};

typedef struct queue Queue;

bool isEmpty(Queue *q){
    return (q->front == -1 || q->front > q->rear);
}

bool isFull(Queue *q){
    return q->rear == q->size -1;
}

void enq(Queue *q , int data){
    if(isFull(q)){
        return ;
    }
    if(q->rear ==-1) q->front++;
    q->rear ++;
}

```

```

        q->arr[q->rear] = data;
    }
    int deq(Queue *q){
        if(isEmpty(q)){
            return -1;
        }
        else{
            q->front++;
            return q->arr[q->front-1];
        }
    }
    bool isVisited(Queue* q , int data){
        for (int i = q->front; i<= q->rear ;i++){
            if(q->arr[i] == data) return true;
        }
        return false;
    }

    void printQ(Queue *q){
        if(isEmpty(q)) {
            printf("The queue is empty\n");
            return;
        }
        printf("\nBFS Traversal of the graph is : ");
        for (int i = q->front ; i<= q->rear ;i++){
            printf("%d " , q->arr[i]);
        }
        printf("\n");
    }
    void printGraph() {
        int i, j;

```

```

printf("\nAdjacency Matrix of graph is :\n");
for (i = 0; i < size; i++) {
    for (j = 0; j < size; j++) {
        printf("%d ", graph[i][j]);
    }
    printf("\n");
}
}

void DFS(int i){
    int j;
    visited[i] = 1;
    printf("%d ",i);
    for(int j =0 ;j< size ;j++){
        if(graph[i][j]==1 && !visited[j]){
            visited[j] = 1;
            DFS(j);
        }
    }
}

void BFS(int i){
    Queue eq,v;
    eq.front = eq.rear = -1;
    v.front = v.rear = -1;
    eq.size = size;
    v.size = size;
    eq.arr = (int*)malloc(eq.size * sizeof(int));
    v.arr = (int*)malloc(v.size * sizeof(int));
    enq(&v , i);
    enq(&eq , i);
    while(!isEmpty(&eq)){
        int u = deq(&eq);

```



```

        for(int j =0 ;j< size ;j++){
            if(graph[u][j] == 1 && !isVisited(&v,j)){
                enq(&v , j);
                enq(&eq , j);
            } } }
    printQ(&v);
}

int main() {
    createNodes();
    printGraph();
    printf("\nDFS traversal of the graph is : ");
    DFS(0);
    BFS(0);

    return 0;
}

```

```

Enter the number of nodes you want to add in the graph : 4
Choose which nodes are connected!
For connected nodes , enter 1 else 0 :
Is node 0 connected to node 0:0
Is node 0 connected to node 1:1
Is node 0 connected to node 2:0
Is node 0 connected to node 3:1
Is node 1 connected to node 1:0
Is node 1 connected to node 2:1
Is node 1 connected to node 3:1
Is node 2 connected to node 0:1
Is node 2 connected to node 2:0
Is node 2 connected to node 3:1
Is node 3 connected to node 3:0

Adjacency Matrix of graph is :
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0

DFS traversal of the graph is : 0 1 2 3
BFS Traversal of the graph is : 0 1 2 3
> 

```

Question 49

WAP to store the graph information in form of adjacency list.

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

struct AdjListNode {
    int dest;
    struct AdjListNode* next;
};

struct AdjList {
    struct AdjListNode *head;
};

struct Graph {
    int V;
    struct AdjList* array;
};

struct AdjListNode* newAdjListNode(int dest) {
    struct AdjListNode* newNode = (struct AdjListNode*) malloc(
        sizeof(struct AdjListNode));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int V) {
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
    graph->V = V;
    graph->array = (struct AdjList*) malloc(V * sizeof(struct AdjList));
    int i;
    for (i = 0; i < V; ++i)
        graph->array[i].head = NULL;

    return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
    struct AdjListNode* newNode = newAdjListNode(dest);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;
    newNode = newAdjListNode(src);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}
```

```

void printGraph(struct Graph* graph) {
    int v;
    for (v = 0; v < graph->V; ++v) {
        struct AdjListNode* pCrawl = graph->array[v].head;
        printf("\n Adjacency list of vertex %d\n head ", v);
        while (pCrawl) {
            printf("-> %d", pCrawl->dest);
            pCrawl = pCrawl->next;
        }
        printf("\n");
    }
}

int main() {
    int V = 5;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);

    printGraph(graph);
    return 0;
}

```

```

Adjacency list of vertex 0
head -> 4-> 1

Adjacency list of vertex 1
head -> 4-> 3-> 2-> 0

Adjacency list of vertex 2
head -> 3-> 1

Adjacency list of vertex 3
head -> 4-> 2-> 1

Adjacency list of vertex 4
head -> 3-> 1-> 0
> □

```