Assignment 1:

To implement addition, multiplication, and transpose of two 2D arrays.

CODE

#include <stdio.h>

void inputArray(int arr[][10],int r,int c){

int i,j;

for (i=0;i<r;i++){

for(j=0; j<c;j++){

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

}

}

}

void showArray(int arr[][10],int r,int c){

int i,j;

for (i=0;i<r;i++){

for(j=0; j<c;j++){

printf("%d ",arr[i][j]);

}

printf("\n");

}

}

void add2DArray(int arr1[][10],int arr2[][10],int arr3[][10],int r1,int c1,int r2,int c2){

int i,j;

for (i=0;i<r1;i++){

for(j=0;j<c1; j++){

arr3[i][j]=arr1[i][j]+arr2[i][j];

}

}

}

void multiply2DArray(int arr1[][10],int arr2[][10],int arr3[][10],int r1,int c1,int r2,int c2){

int i,j,k;

for (i=0;i<r1;i++){

for(j=0;j<c1; j++){

arr3[i][j]=0;

for(k=0;k<c1;k++){

arr3[i][j]+=(arr1[k][j]\*arr2[i][k]);

}

}

}

}

void transposeMatrix(int arr[][10],int transpose[][10],int r,int c){

int i,j;

for(i=0; i<r; i++){

for(j=0;j<c; j++){

transpose[i][j]=arr[j][i];

}

}

}

int main() {

int arr1[10][10],arr2[10][10],arr3[10][10],transpose[10][10];

int r1,c1,r2,c2;

//Enter Rows and Columns of 2D-Arrays

printf("\n Enter the Rows and Columns of First 2D-Array :\n");

scanf("%d%d",&r1,&c1);

printf("\n Enter the Rows and Columns of Second 2D-Array :\n");

scanf("%d%d",&r2,&c2);

//Enter the Array elements

printf("\nEnter the elemnts of First 2D-Array :\n");

inputArray(arr1,r1,c1);

printf("\nEnter the elemnts of Second 2D-Array :\n");

inputArray(arr2,r2,c2);

//Print the 2D-Arrays

printf("\nFirst 2D-Array is :\n");

showArray(arr1,r1,c1);

printf("\nSecond 2D-Array is :\n");

showArray(arr2,r2,c2);

//Addition of Two 2D-Arrays

if (r1!=r2 || c1 !=c2){

printf("\nAddition of these 2-D Array is not possible\n");

}

else{

printf("\nAddition of two 2D-Arrays are\n");

add2DArray(arr1,arr2,arr3,r1,c1,r2,c2);

showArray(arr3,r1,c1);

}

//Multiplication of two 2-D Arrays

if(c1!=r2){

printf("\nMatrix Multiplication is not possible");

}

else{

printf("\nMatrix Multiplication is :\n");

multiply2DArray(arr1,arr2,arr3,r1,c1,r2,c2);

showArray(arr3,r1,c1);

}

//Transpose of the 2-D Array is

printf("\nTranspose of the First 2D-Array is :\n");

transposeMatrix(arr1,transpose,r1,c1);

showArray(transpose,r1,c1);

printf("\nTranspose of the Second 2D-Array is :\n");

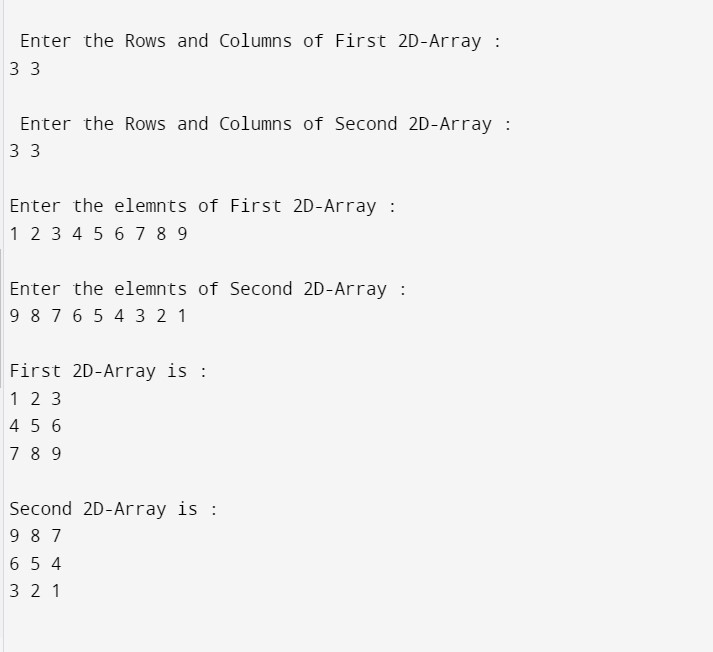
transposeMatrix(arr2,transpose,r2,c2);

showArray(transpose,r2,c2);

return 0;

}

OUTPUT:



A white screen with black text

Description automatically generated

Assignment 2:

To implement Linear Search and Binary Search

CODE:

#include<stdio.h>

void input\_array(int arr[10],int size)

{

int i;

for(i=0; i<size; i++)

{

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

}

}

void show\_array(int arr[10],int size)

{

int i;

for(i=0; i<size; i++)

{

printf("%d ",arr[i]);

}

}

int linear\_search(int arr[10], int size, int ele)

{

int i;

for (i=0; i<size; i++)

{

if (arr[i]==ele)

{

return i;

}

}

return -1;

}

int binary\_search(int arr[10], int size, int ele)

{

int mid, beg=0, end=size-1;

while(beg<=end)

{

mid = (beg + end)/2;

if(arr[mid]==ele)

{

return mid;

}

else if(arr[mid] > ele)

{

end= mid-1;

}

else

{

beg= mid +1;

}

}

return -1;

}

int main()

{

int arr[10], size, k, ele, choice;

printf("\nEnter the size of Array : ");

scanf("%d",&size);

if(size>10)

{

printf("\n Size is Out of Range");

}

else

{

printf("\nEnter the elements of array :\n");

input\_array(arr,size);

printf("\nEntered elements are :\n");

show\_array(arr,size);

printf("\nEnter the element to be searched : ");

scanf("%d",&ele);

printf("\nEnter the Choice 1 :- For Linear Search, 2 :- For Binary Search: ");

scanf("%d",&choice);

switch (choice)

{

case 1:

printf("\nSearching the element through Linear Search :\n");

k=linear\_search(arr,size,ele);

if (k==-1)

{

printf("\n Element Not Found");

}

else

{

printf("\nElement Found at the %d index ",k);

}

break;

case 2:

printf("\nSearching the element through Binary Search :\n");

k=binary\_search(arr,size,ele);

if (k==-1)

{

printf("\nElement Not Found");

}

else

{

printf("\nElement Found at %d index",k);

}

break;

default:

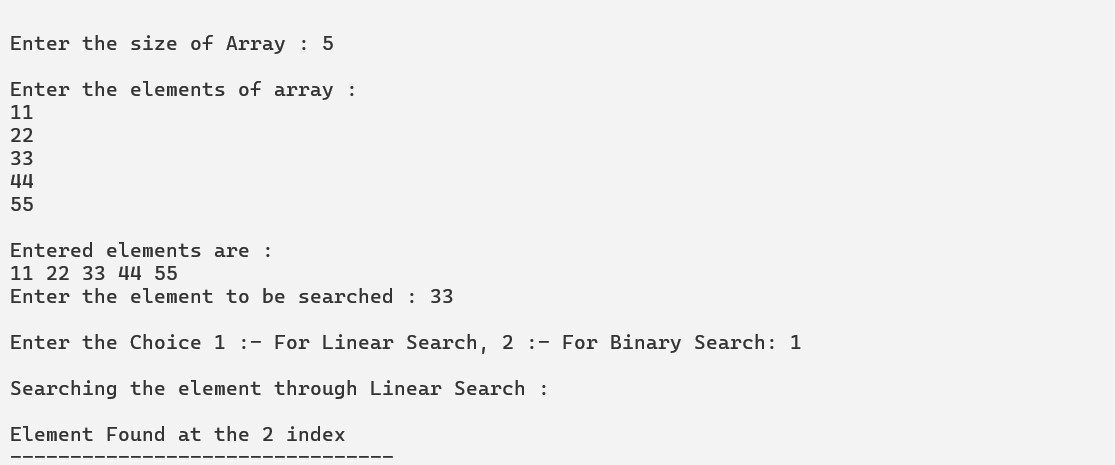
printf("\n Inavlid Choice");

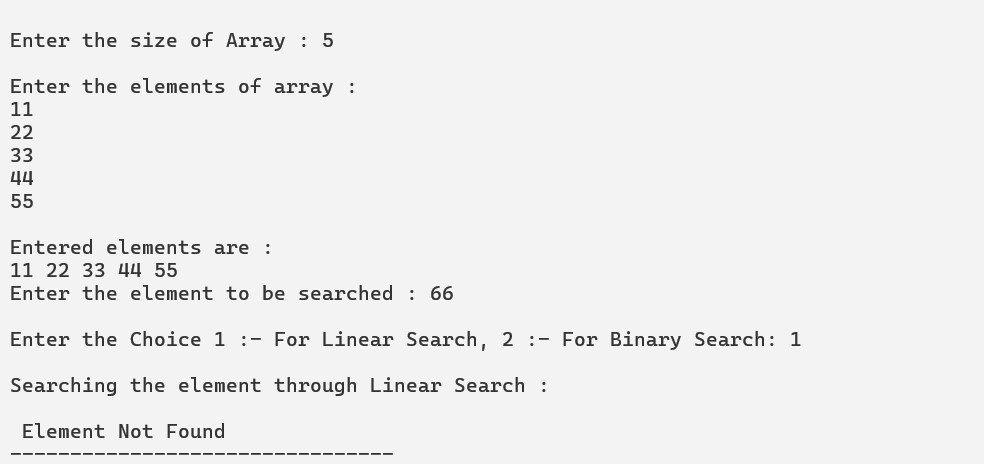
}

}

}

OUTPUT:





A white screen with black text

Description automatically generated

A white screen with black text

Description automatically generated

**Assignment 3:**

To implement stack, queue and circular queue using array.

1. STACK IMPLEMENTATION USING ARRAY

CODE:

1. Stack Implementation using array

#include <stdio.h>

# define MAX\_SIZE 5

#include <stdlib.h>

int isFull(int \*top)

{

if ((\*top)==MAX\_SIZE-1)

{

return 1;

}

else

{

return 0;

}

}

int isEmpty(int \*top)

{

if ((\*top)==-1)

{

return 1;

}

return 0;

}

void push (int array[10], int \*top, int val)

{

if (isFull(top))

{

printf("\n Stack Overflow....");

}

else

{

(\*top)++;

array[\*top]=val;

}

}

int pop(int array[10],int \*top)

{ int val;

if (isEmpty(top))

return -1;

else

{

val= array[\*top];

(\*top)--;

return val;

}

}

void display(int array[10], int top)

{

int i;

if (top==-1)

printf("\nStack is Empty ");

else

for (i=top; i>=0; i--)

printf("array[%d]: %d \n",i,array[i]);

}

int main()

{

int array[20],val, top=-1, size, i, choice, k, num, del;

for (;;)

{

printf("\n1: Add the Elements in Stack.");

printf("\n2. Delete the Elements from Stack.");

printf("\n3. Display All Elements of Stack.");

printf("\n4. Exit");

printf("\n Enter the Choice : ");

scanf("%d",&choice);

switch (choice)

{

case 1:

printf("\nEnter the Value to be Inserted : ");

scanf("%d",&val);

push (array, &top, val);

break;

case 2:

del=pop(array,&top);

if (del!=-1)

printf("\nDeleted Number is :%d",del);

else

printf("\nStack Underflow");

break;

case 3:

display(array,top);

break;

case 4:

exit(0);

break;

default:

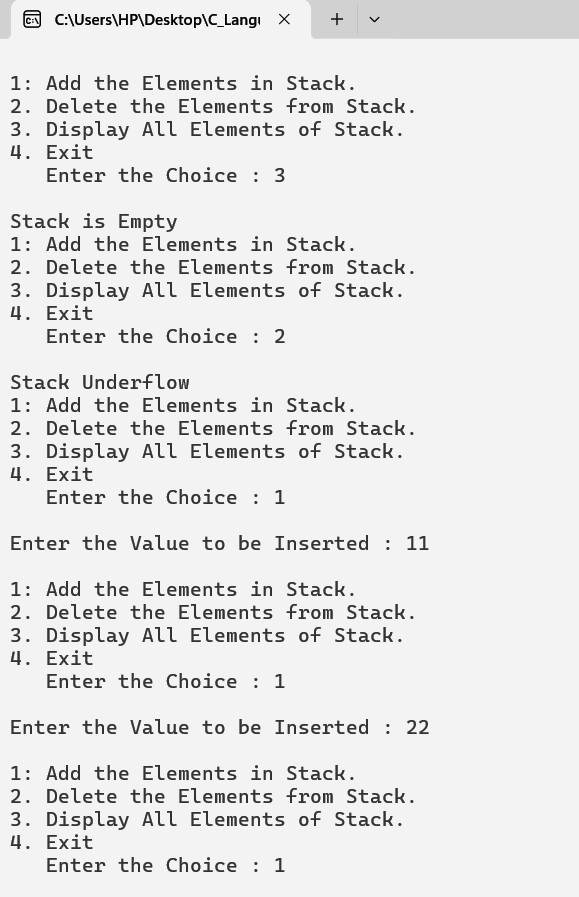
printf("\nInvalid Choice ...");

}

}

return 0;

}

**OUTPUT:**

A screenshot of a computer program

Description automatically generated

**Assignment 3:**

To implement stack, queue and circular queue using array.

1. QUEUE IMPLEMENTATION USING ARRAY

CODE:

#include<stdio.h>

#include<stdlib.h>

#define MAX\_SIZE 5

int isFull(int array[MAX\_SIZE],int \*rear)

{

if((\*rear)==MAX\_SIZE-1)

{

return 1;

}

return 0;

}

void enqueue(int array[MAX\_SIZE],int \*front, int \*rear,int val)

{

if(isFull(array,rear))

{

printf("\nQueue is Full");

}

else

{

(\*rear)++;

array[\*rear]=val;

if((\*front)== -1)

{

(\*front)++;

}

}

}

int isEmpty(int array[MAX\_SIZE],int \*front)

{

if ((\*front)==-1)

{

return 1;

}

return 0;

}

int dequeue(int array[MAX\_SIZE],int \*front, int \*rear)

{

int val;

if(isEmpty(array,front))

return -1;

else

{

val=array[\*front];

(\*front)++;

if (\*front > \*rear)

{

\*front = \*rear =-1;

}

return val;

}

}

void display(int array[MAX\_SIZE],int \*front, int \*rear)

{

int i;

if (\*front==-1 || \*front > \*rear)

{

printf("\n Queue is Empty");

}

else

{

for (i=\*front; i<=\*rear;i++)

{

printf("%d ",array[i]);

}

}

}

int main()

{

int array[MAX\_SIZE],front=-1,rear=-1,k,val,choice,del;

for(;;)

{

printf("\n1:Enqueue elements in Queue.");

printf("\n2:Dequeue elements from Queue.");

printf("\n3:Display Queue.");

printf("\n4:Exit");

printf("\nEnter the Choice :");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("\nEnter the element to be enqueued : ");

scanf("%d",&val);

enqueue(array,&front,&rear,val);

break;

case 2:

del=dequeue(array,&front,&rear);

if(del!=-1)

printf("\nDequeued Element is %d",del);

else

printf("\nQueue Underflow");

break;

case 3:

display(array,&front,&rear);

break;

case 4:

exit(0);

default:

printf("\nInvalid Choice");

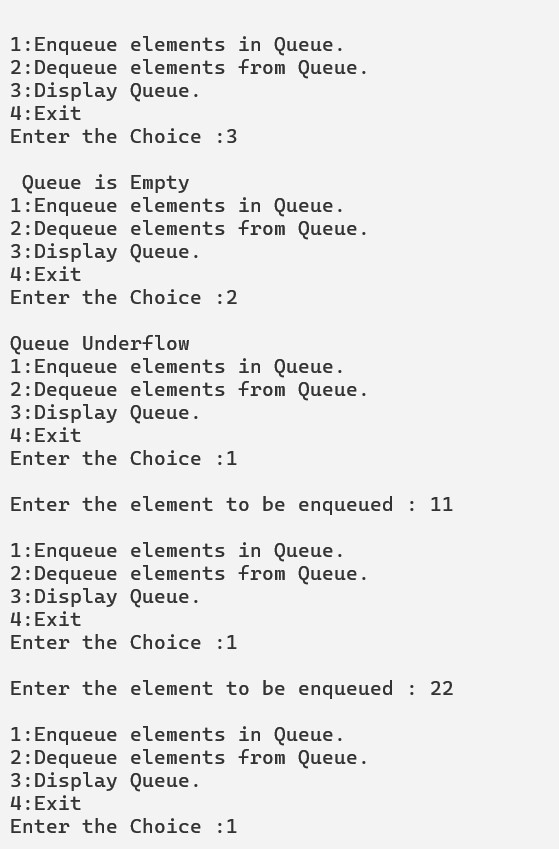
}

}

return 0;

}

**OUTPUT:**



A screenshot of a computer program

Description automatically generated

**Assignment 3:**

To implement stack, queue and circular queue using array.

1. CIRCULAR QUEUE IMPLEMENTATION USING ARRAY

CODE:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 5

int isFull(int circularQueue[], int \*rear, int \*front)

{

if ((((\*rear)+1)%MAX\_SIZE)==(\*front))

{

return 1;

}

return 0;

}

void enqueueCircularQueue(int circularQueue[],int \*front, int \*rear, int val)

{

if(isFull(circularQueue,rear,front))

{

printf("\n Queue is Full");

}

else

{

\*rear= (\*rear+1)%MAX\_SIZE;

circularQueue[\*rear]=val;

if(\*front==-1)

{

(\*front)++;

}

printf("Front and Rear at Enqueue %d , %d\n",\*front,\*rear);

}

}

int isEmpty(int circularQueue[],int \*front, int \*rear)

{

if (\*front == -1)

{

return -1;

}

return 0;

}

int dequeue(int circularQueue[], int \*front, int \*rear)

{

int val;

if (isEmpty(circularQueue,front,rear))

{

return -1;

}

else

{

val = circularQueue[\*front];

\*front = (\*front + 1)%MAX\_SIZE;

if(\*front == \*rear)

{

\*front =\*rear =-1;

}

printf("Front and Rear at Enqueue %d , %d\n",\*front,\*rear);

return val;

}

}

void display(int circularQueue[], int \*front, int \*rear)

{

int i;

if(\*front == 0 && \*rear==0)

{

printf("\n Queue is Empty");

}

else

{

for (i=\*front; i<=\*rear; i++)

{

printf("%d ",circularQueue[i]);

}

}

}

int main()

{

int circularQueue[MAX\_SIZE],front=0, rear=0,choice, val,del;

for(;;)

{

printf("\n1: Enqueue element in Circular Queue");

printf("\n2: Dequeue element from Circular Queue");

printf("\n3: Display All ");

printf("\n4: Exit");

printf("\n Enter Your Choice :");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("\nEnter the Element to be enqueued : ");

scanf("%d",&val);

enqueueCircularQueue(circularQueue, &front, &rear, val);

break;

case 2:

del=dequeue (circularQueue, &front, &rear);

if (del ==-1)

{

printf("\n Queue Underflow");

}

else

printf("\nDeleted Element is: %d",del);

break;

case 3:

display (circularQueue, &front, &rear);

break;

case 4:

exit (0);

default:

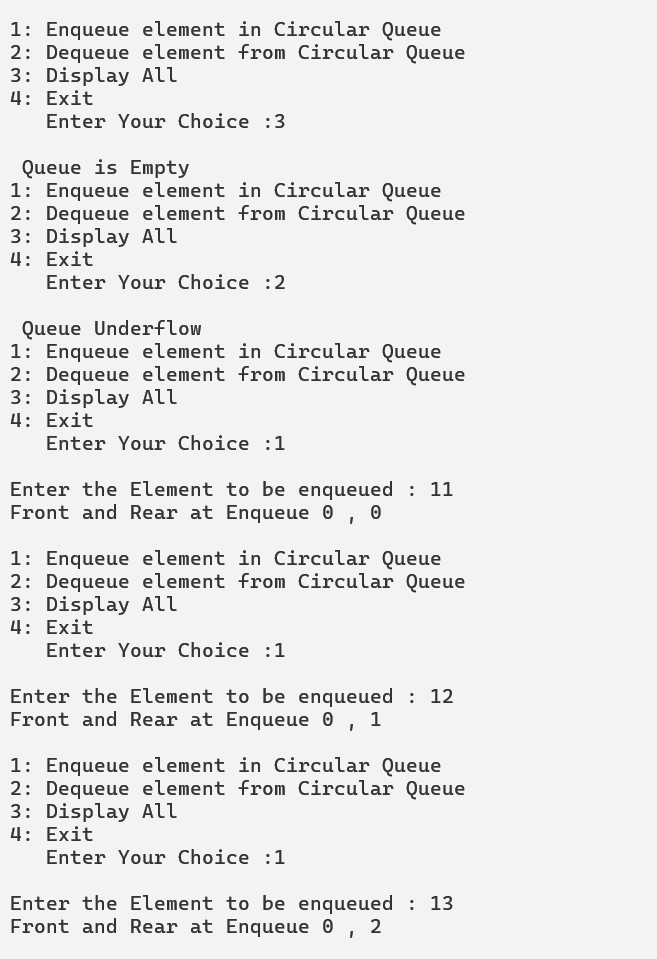
printf("\nInvalid Choice");

}

}

return 0; }

**OUTPUT:-**



A screenshot of a computer program

Description automatically generated

**Assignment 4:**

To implement insert, and deletion operations in Linked List.

**CODE:**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

//Insert Element at Beg of Node.

void insertBeg(struct node \*\* head, int val)

{

struct node \*temp;

temp = (struct node \*)malloc(sizeof(struct node));

temp->data = val;

temp->next =(\*head); //|10 | |20 | |30 | |40 | |50 |

(\*head)= temp;

printf("\n Value Inserted Successfully .");

}

//Insert Element in List at the End.

void insertEnd(struct node \*\*head, int val)

{

struct node \*temp;

struct node \*t;

temp = (struct node \*)malloc(sizeof(struct node));

temp->data= val;

temp->next = NULL;

t = (\*head);

while ((t->next) !=NULL)

{

printf("1 ");

t = t->next;

}

t->next = temp;

printf("\n Value Inserted Successfully .");

}

//Insert at Particular Position.

void insertPos(struct node \*\*head, int val, int k)

{

int i;//|10 add| |20 add| |30 add | |40 add| |50 add|

struct node \*temp, \*ptr;

ptr = (\*head);

temp = (struct node \*)malloc(sizeof(struct node));

temp->data=val;

for (i=1; i<k-1; i++)

{

ptr= ptr->next;

}

temp->next = ptr->next;

ptr->next = temp;

printf("\n Value Inserted Successfully .");

}

//Delete from the Beginning

void deleteBeg(struct node \*\*head)

{

if ((\*head)==NULL)

{

return -1;

}

struct node \*temp = \*head;

temp = (struct node \*)malloc(sizeof(struct node));

printf("%d ",(\*head)->data);

temp = \*head;

printf("Data is Deleted %d",temp ->data);

\*head = temp -> next;

free(temp);

printf("\n Data Deleted Sucessfully ");

}

//Delete from the End

void deleteEnd(struct node \*\*head)

{

if ((\*head)==NULL)

{

return -1;

}

struct node \*temp = \*head, \*prev;

temp = (struct node \*)malloc(sizeof(struct node));

temp = \*head;

prev = \*head;

while ((temp -> next))

{

prev = temp;

temp = temp->next;

}

prev->next =NULL;

printf("Data is Deleted %d",temp ->data);

free(temp);

printf("\n Data Deleted Sucessfully ");

}

//Delete from the Specified Position

int deletePos(struct node \*\*head, int val)

{

struct node \*temp, \*prev;

temp=(struct node \*)malloc(sizeof(struct node));

temp = \*head;

prev = \*head;

while ((temp->data)!=val)

{

prev = temp;

temp = temp->next;

}

if ((temp->data) != val)

{

return -1;

}

else

{

prev->next = temp->next;

printf("\nDeleted Element is %d",temp->data);

free(temp);

}

}

void display(struct node \*head)

{

while (head != NULL)

{

printf("%d ->",head->data);

head = head->next;

}

printf(" NULL\n");

}

int main()

{

struct node \*head = NULL;

int val,k, choice,ch;

for (;;)

{

printf("\n------------------------------------------------\n");

printf("\n1: Insertion in Linked List at the Begining ");

printf("\n2: Insertion in Linked List at the End ");

printf("\n3: Insertion in Linked List at the Specified Position ");

printf("\n4: Delete the Linked List from Beginning ");

printf("\n5: Delete the Linked List from End ");

printf("\n6: Delete the Linked List at the Specified Value ");

printf("\n7: Display");

printf("\n8: Exit");

printf("\n Enter Your Choice :");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("\n Enter the value to be inserted :");

scanf("%d",&val);

insertBeg (&head, val);

break;

case 2:

printf("\n Enter the value to be inserted :");

scanf("%d",&val);

insertEnd (&head, val);

break;

case 3:

printf("\n Enter the value to be inserted :");

scanf("%d",&val);

printf("\nEnter Position of Insertion :");

scanf("%d",&k);

insertPos(&head, val, k);

break;

case 4:

k=deleteBeg(&head);

if (k==-1)

{

printf("\nList is Empty");

}

break;

case 5:

k=deleteEnd(&head);

if (k==-1)

{

printf("\nList is Empty");

}

break;

case 6:

printf("\nEnter the Value to be Deleted ");

scanf("%d",&val);

ch=deletePos(&head,val);

if (ch==-1)

{

printf("\nElement Not Found ...");

}

break;

case 7:

display(head);

break;

case 8:

exit(0);

default:

printf("\nInvalid Choice...");

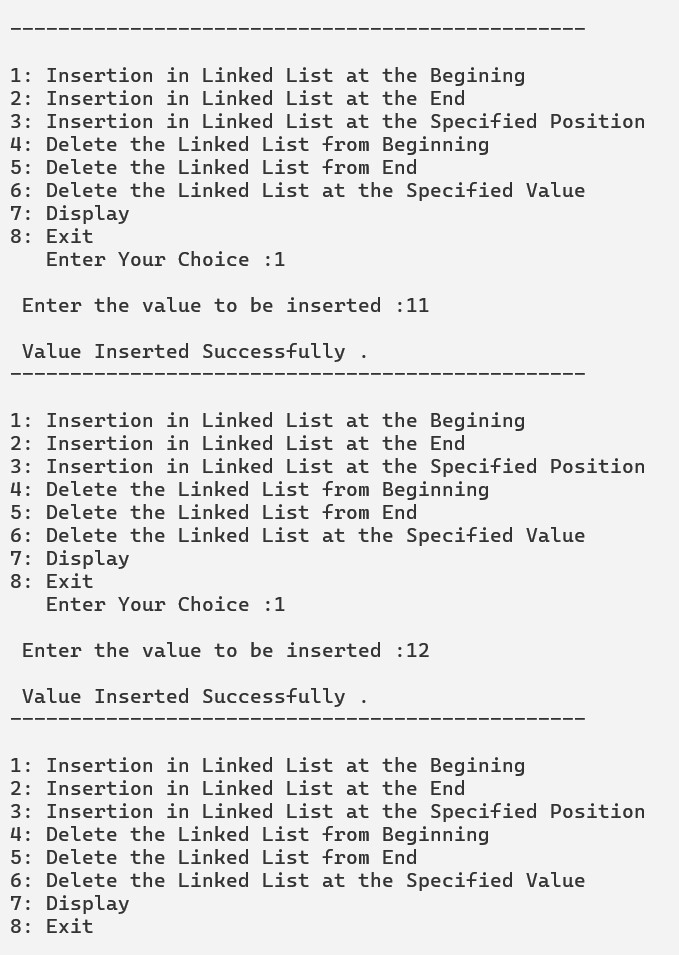
}

}

return 0;

}

**OUTPUT:**



A white paper with black text

Description automatically generated

A screenshot of a computer program

Description automatically generated

A white paper with black text

Description automatically generated

A white paper with black text

Description automatically generated

A screenshot of a computer program

Description automatically generated

A white paper with black text

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Assignment 5.1:**

**To implement Stack Using Linked List**

**CODE: -**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

};

void push (struct node \*\*topAdd, int val)

{

struct node \*temp;

temp = (struct node \*)malloc(sizeof(struct node));

temp->data = val;

temp->next =\*topAdd;

\*(topAdd) = temp;

}

int pop(struct node \*\*topAdd)

{

int val;

struct node \*temp;

if (\*topAdd == NULL)

{

return -1;

}

else

{

temp = (struct node \*)malloc(sizeof(struct node));

temp = \*topAdd;

val= temp->data;

(\*topAdd) = (\*topAdd)->next;

free(temp);

return val;

}

}

void display(struct node \*top)

{

if (top==NULL)

{

printf("\n Stack is Empty");

}

else

{

do

{

printf("%d ",top->data);

top = top->next;

}

while (top!=NULL);

}

}

int main()

{

struct node \*top=NULL;

int choice,k,val;

for (;;)

{

printf("\nStack Implementation using Linked List ...\n");

printf("\n1: Push");

printf("\n2: Pop");

printf("\n3: Display");

printf("\n4: Exit");

printf("\nEnter Your Choice :");

scanf("%d",&choice);

switch (choice)

{

case 1:

printf("\nEnter Value To be Inserted :");

scanf("%d",&val);

push(&top, val);

break;

case 2:

k=pop(&top);

if (k!=-1)

printf("\nDeleted Element is %d",k);

else

printf("\nStack Underflow");

break;

case 3:

display(top);

break;

case 4:

exit(0);

default:

printf("Invalid Choice...");

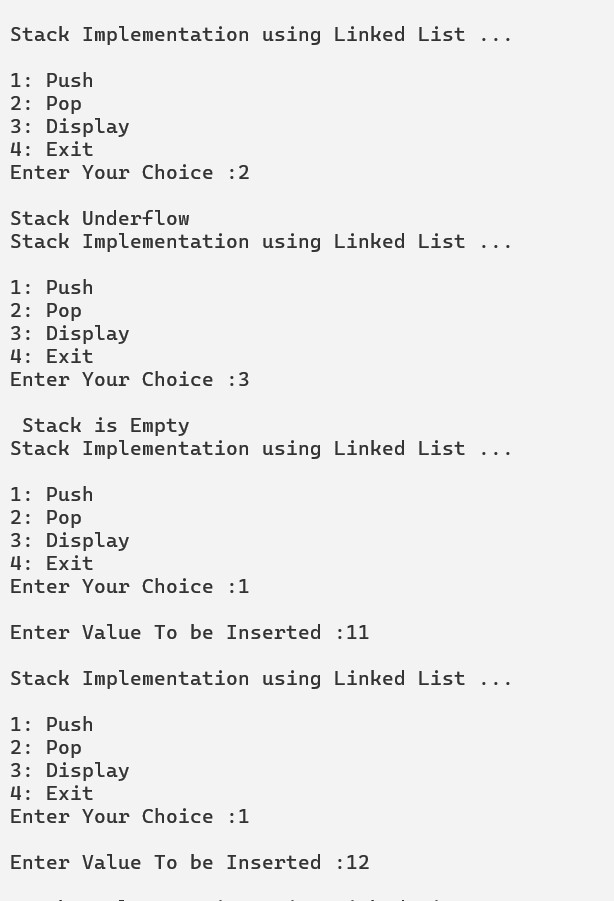
}

}

return 0;

}

**OUTPUT:**



A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Assignment 5.2:**

**To implement Queue Using Linked List**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

};

void enqueue(struct node \*\*front, struct node \*\*rear, int val)

{

struct node \*temp;

temp = (struct node \*)malloc(sizeof(struct node));

temp->data = val;

temp->next= NULL;

if (\*rear == NULL)

{

\*front = temp;

\*rear = temp;

}

else

{

(\*rear)->next = temp;

\*rear = temp;

}

}

int dequeue(struct node \*\*front, struct node \*\*rear)

{

int val;

struct node \*temp;

temp = (struct node \*)malloc(sizeof(struct node));

if (\*front == NULL)

{

return -1;

}

else

{

temp= \*front;

val = temp ->data;

(\*front)= temp->next;

if (\*front == NULL)

{

\*rear = NULL ;

}

free(temp);

return val;

}

}

void display(struct node \*front, struct node \*rear)

{

if (front == NULL)

{

printf("\nQueue is Empty");

return;

}

else

{

do{

printf("%d ->",front->data);

front =front->next;

}while(front!=NULL);

printf("NULL");

}

}

int main()

{

struct node \*front = NULL, \*rear = NULL;

int choice,k,val;

for (;;)

{

printf("\nQueue Implementation using Linked List ...\n");

printf("\n1: Enqueue");

printf("\n2: Dequeue");

printf("\n3: Display");

printf("\n4: Exit");

printf("\nEnter Your Choice :");

scanf("%d",&choice);

switch (choice)

{

case 1:

printf("\nEnter Value To be Inserted :");

scanf("%d",&val);

enqueue(&front,&rear, val);

break;

case 2:

k=dequeue(&front, &rear);

if (k!=-1)

printf("\nDeleted Element is %d",k);

else

printf("\nQueue Underflow");

break;

case 3:

display(front, rear);

break;

case 4:

exit(0);

default:

printf("Invalid Choice...");

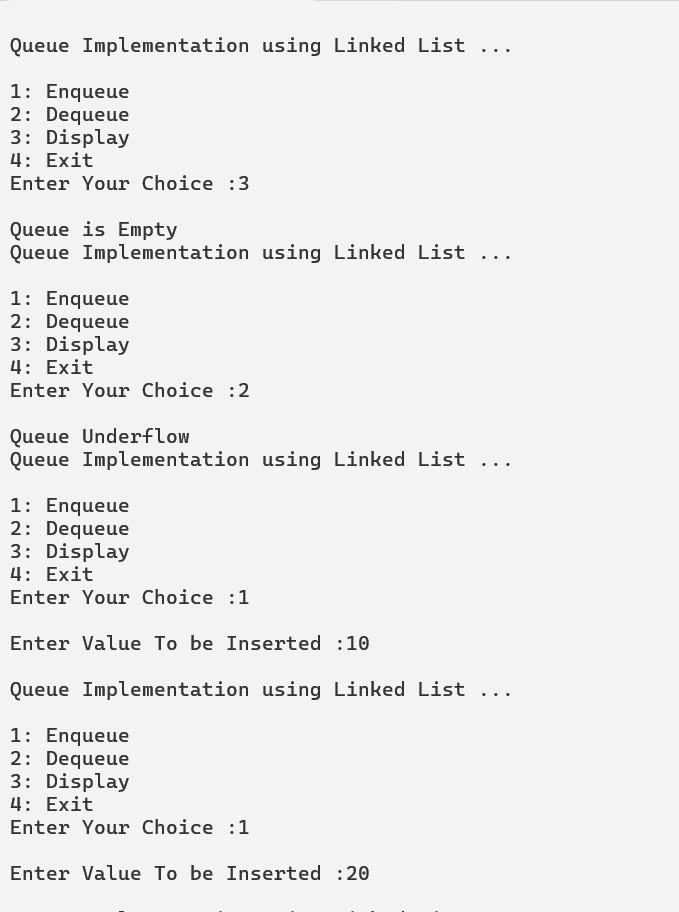
}

}

return 0;

}

**OUTPUT:**



A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Assignment 6.1:**

**To implement Selection Sort Using Array**

**CODE:**

#include <stdio.h>

void input(int arr[10], int size)

{

int i;

for (i=0; i<size; i++)

{

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

}

}

void show(int arr[10], int size)

{

int i;

for (i=0; i<size; i++)

{

printf("%d ",arr[i]);

}

printf("\n");

}

void selsort(int arr[10], int size)

{

int i,j,key,temp;

for (i=0; i<size; i++)

{

key=i;

for (j=i+1; j<size; j++)

{

if (arr[j] < arr[key])

{

key=j;

}

}

if(key!=i)

{

temp =arr[key];

arr[key]=arr[i];

arr[i]= temp;

}

printf("\nArray after %d duration ",i);

show(arr,size);

}

}

int main()

{

int arr[10],size;

printf("\nEnter size of array :");

scanf("%d",&size);

printf("\nEnter Elements of array :\n");

input(arr,size);

printf("\nEntered Elements are :\n");

show(arr,size);

selsort(arr,size);

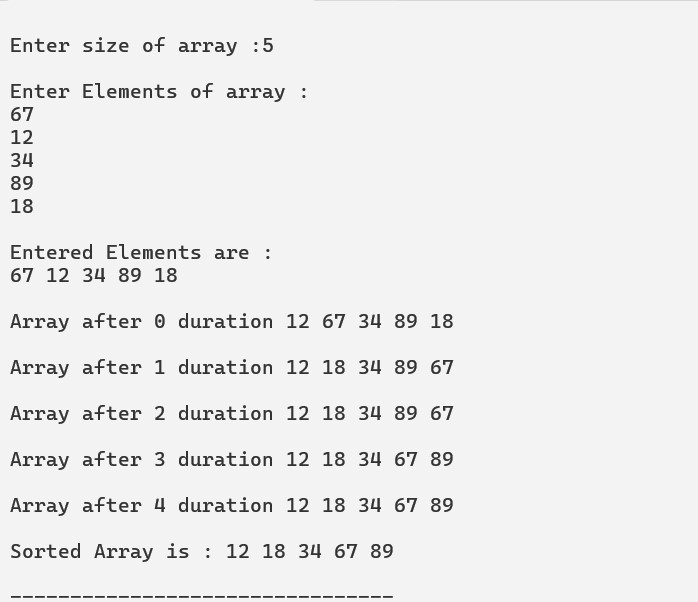
printf("\nSorted Array is : ");

show(arr,size);

return 0;

}

**OUTPUT:**



**Assignment 6.2:**

**To implement Bubble Sort Using Array**

**CODE:**

#include <stdio.h>

void input(int arr[10], int size)

{

int i;

for (i=0; i<size; i++)

{

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

}

}

void show(int arr[10], int size)

{

int i;

for (i=0; i<size; i++)

{

printf("%d ",arr[i]);

}

printf("\n");

}

void bubblesort(int arr[10], int size)

{

int i,j,temp;

for (i=0; i<size-1; i++)

{

for (j=0; j<size-1-i; j++)

{

if (arr[j]>arr[j+1])

{

temp = arr[j];

arr[j]= arr[j+1];

arr[j+1]= temp;

}

}

printf("\nArray after %d duration ",i);

show(arr,size);

}

}

int main()

{

int arr[10],size;

printf("\nEnter size of array :");

scanf("%d",&size);

printf("\nEnter Elements of array :\n");

input(arr,size);

printf("\nEntered Elements are :\n");

show(arr,size);

bubblesort(arr,size);

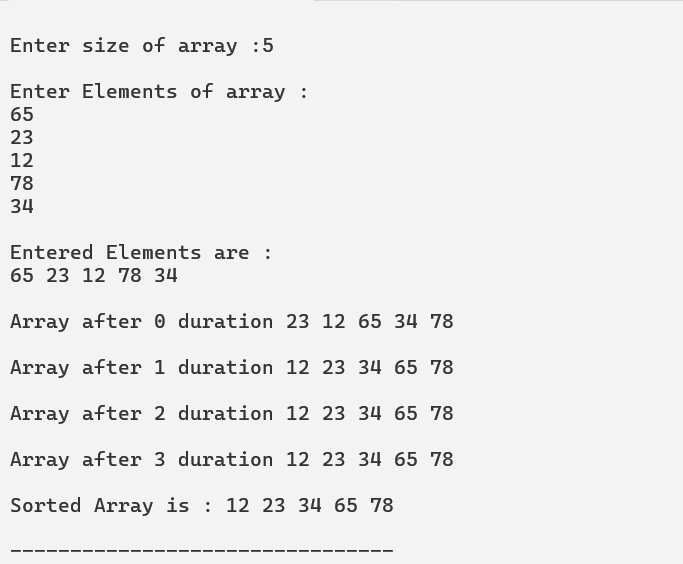
printf("\nSorted Array is : ");

show(arr,size);

return 0;

}

**OUTPUT:**



**Assignment 6.3:**

**To implement Insertion Sort Using Array**

**CODE:**

#include <stdio.h>

void input(int arr[10], int size)

{

int i;

for (i=0; i<size; i++)

{

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

}

}

void show(int arr[10], int size)

{

int i;

for (i=0; i<size; i++)

{

printf("%d ",arr[i]);

}

printf("\n");

}

void insertSort(int arr[10], int size)

{

int key,i,j;

for (i=1; i<=size-1; i++)

{

key = arr[i];

j = i-1;

while (j>=0 && arr[j]>key){

arr[j+1]=arr[j];

j--;

}

arr[j+1]=key;

printf("\nArray after %d duration ",i);

show(arr,size);

}

}

int main()

{

int arr[10],size;

printf("\nEnter size of array :");

scanf("%d",&size);

printf("\nEnter Elements of array :\n");

input(arr,size);

printf("\nEntered Elements are :\n");

show(arr,size);

insertSort(arr,size);

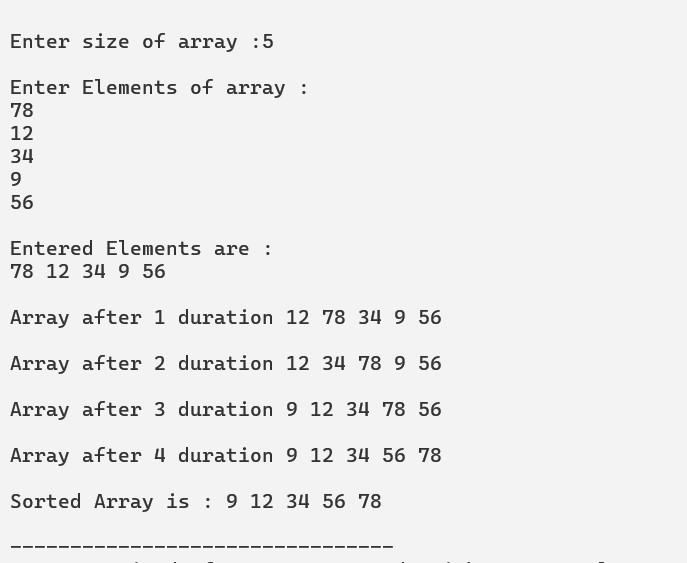
printf("\nSorted Array is : ");

show(arr,size);

return 0;

}

**OUTPUT:**



**ASSIGNMENT 7:**

**TO IMPLEMENT A GRAPH**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

struct adj {

int node\_no;

struct node \*next;

};

void insert(struct adj \*new\_list[], int node, int nb){

struct adj \*temp=NULL;

int a,i;

temp = (struct adj \*)malloc(sizeof(struct adj));

if (!temp)

{

printf("\n Memory Allocation failed ...");

exit(1);

}

printf("\nEnter the value of Neighbour/ Neighbour No for node %d:",node);

scanf("%d",&a);

temp->node\_no = a;

temp ->next =NULL;

new\_list[node] = temp;

for (i=1; i<nb; i++)

{

struct adj \*temp1 = NULL;

temp1 = (struct adj \*)malloc(sizeof(struct adj));

printf("\nEnter next Neighbour No :");

scanf("%d",&a);

temp1 ->node\_no = a;

temp1 ->next =NULL;

temp->next= temp1;

temp =temp1;

}

}

void display(struct adj \*L[], int n)

{

int i,j;

struct adj \*temp;

printf("\n The Graph is :\n");

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

{

printf("\n Current Node is :%d ",i);

if (L[i] == NULL)

{

printf("\nVertex has no neighbour node");

}

else

{

temp = L[i];

printf("\n Neighbours are :");

j=0;

while(temp !=NULL)

{

printf("%d ->",temp->node\_no);

j++;

temp = temp->next;

}

printf("NULL");

printf("\n it has %d neighbours ",j);

}

printf("\n");

}

}

int main()

{

struct adj \*\*G;

int n,i,nb; //n= number of vertices, nb = number of neighbours to a vertex

printf("Enter the number of vertex in a graph :");

scanf("%d",&n);

G = (struct adj \*\*)malloc(n \* sizeof(struct adj \*));

if (!G)

{

printf("\nMemeory Allocation failed...");

return 1;

}

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

{

G[i] =NULL;

printf("\n Enter the number of neighbour nodes of %d vertex:",i);

scanf("%d",&nb);

if (nb > 0)

{

insert(G, i, nb);

}

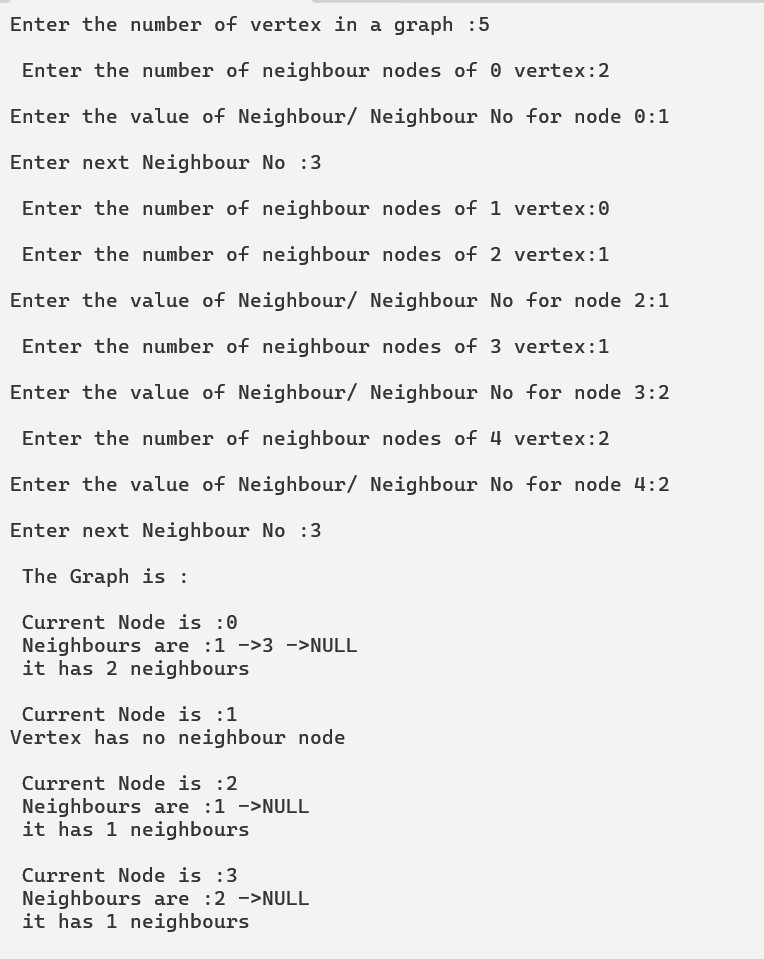
}

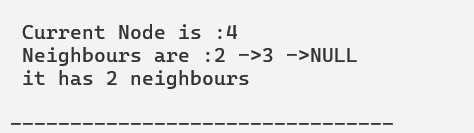
display(G,n);

return 0;

}

**OUTPUT:**





**ASSIGNMENT 8:**

**TO IMPLEMENT A BFS USING LINKED LIST**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

struct adj {

int nodeno;

struct adj \*next;

};

void insert (struct adj \*G[], int node, int nb)

{

struct adj \*temp=NULL;

int a,i;

temp = (struct adj \*)malloc(sizeof(struct adj));

if (!temp)

{

printf("\n Memory Allocation failed ...");

exit(1);

}

printf("\nEnter the value of Neighbour/ Neighbour No for node %d:",node);

scanf("%d",&a);

temp->nodeno = a;

temp ->next =NULL;

G[node] = temp;

for (i=1; i<nb; i++)

{

struct adj \*temp1 = NULL;

temp1 = (struct adj \*)malloc(sizeof(struct adj));

printf("\nEnter next Neighbour No :");

scanf("%d",&a);

temp1 ->nodeno = a;

temp1 ->next =NULL;

temp->next= temp1;

temp =temp1;

}

}

void display(struct adj \*L[], int n)

{

int i,j;

struct adj \*temp;

printf("\n The Graph is :\n");

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

{

printf("\n Current Node is :%d ",i);

if (L[i] == NULL)

{

printf("\nVertex has no neighbour node");

}

else

{

temp = L[i];

printf("\n Neighbours are :");

j=0;

while(temp !=NULL)

{

printf("%d ->",temp->nodeno);

j++;

temp = temp->next;

}

printf("NULL");

printf("\n it has %d neighbours ",j);

}

printf("\n");

}

}

void insertq(struct adj \*\*front, struct adj \*\*rear, int item)

{

struct adj \*temp = NULL;

temp = (struct adj \*)malloc(sizeof(struct adj));

if (\*rear ==NULL)

\*rear = \*front = temp;

else

{

(\*rear)->next = temp;

(\*rear) =temp;

}

(\*rear)->nodeno = item;

(\*rear)->next = NULL;

printf("\n Inserted item is = %d",item);

}

int deleteq(struct adj \*\*front, struct adj \*\*rear)

{

struct adj \*temp;

int a;

if (\*front == NULL)

return -1;

temp = \*front;

a = (\*front)->nodeno;

if ((\*front)==(\*rear))

{

(\*front) = (\*rear) = NULL;

}

else{

(\*front) = (\*front)->next;

}

free(temp);

return a;

}

void BFS(struct adj \*G[], int n, int s)

{

int i, status[10], node;

struct adj \*front = NULL;

struct adj \*rear = NULL;

struct adj \*temp = NULL;

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

{

status[i] =1;

}

insertq(&front, &rear, s);

status[s]=2;

while (front != NULL)

{

node = deleteq(&front, &rear);

status[node] =3;

printf("\n Deleted :");

printf(" %d",node);

temp = G[node];

while (temp != NULL)

{

if (status[temp->nodeno]==1)

{

insertq (&front, &rear, temp->nodeno);

status[temp->nodeno]=2;

}

temp = temp->next;

}

}

}

int main()

{

struct adj \*\*G;

int n,i,nb;

printf("\nEnter the number of nodes :");

scanf("%d",&n);

G = (struct adj \*\*)malloc(n \* sizeof(struct adj \*));

if (!G)

{

printf("\n Memory Allocation failed..");

return ;

}

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

{

G[i]=NULL;

printf("\n Enter the number of neighbour nodes.. of %d vertex ",i);

scanf("%d",&nb);

if (nb >0)

{

insert(G, i, nb);

}

}

display(G, n);

printf("\n BFS Implementation is :");

BFS(G, n, 0);

return 0;

}

**OUTPUT:**

