**Lingaya’s Vidyapeeth, Faridabad**

(Deemed to be University under Section 3 of UGC Act, 1956)

**Data Structures using C Lab**

**CS-153**

**LAB File**

**B.TECH 2nd Year [C.S.E]**



Nachauli, Old Faridabad-Jasana Road,

Faridabad 121002, Phone: (129) 2201008, 2201009;

WebSite: www.lingayasuniversity.edu.in

**Submitted to: Submitted By:**

Dr. Tapsi NAME – S Kashaf Alam

ROLL No – 22CS75CL

SEMESTER – 3rd

**Index**

|  |  |
| --- | --- |
| **S.No.** | **Program Name** |
| **A** | **ARRAY OPERATIONS** |
| 1 | Write a program to insert an element at given position in linear array |
| 2 | Write a program to insert an element in sorted array |
| 3 | Write a program to delete an element from given position in linear array |
| 4 | Perform following operations on matrices using functions only:   1. Addition ii) Subtraction iii) Multiplication iv) Transpose |
|  |  |
| **B** | **SEARCHING** |
| 5 | Write a program to search an element in an array using linear search. |
| 6 | Using iteration & recursion concepts write programs for finding the element in the array using Binary Search. |
|  |  |
| C | **RECURSION** |
| 7 | Write a program to compute factorial of given number using recursion |
| 8 | Write as program to solve Tower of Hanoi problem using recursion |
| 9 | Write a program to find power of given number using recursion |
|  |  |
| **D** | **STACK & QUEUE** |
| 10 | Write a program for static implementation of Stack |
| 11. | Write a program for dynamic implementation of queue |
| 12 | Write a program for static implementation of circular Queue. |
| 13 | Write a program to evaluate a postfix operation |
|  |  |
| **E** | **LINKED LISTS** |
| 14 | Write a program to create a linked list & perform operations such as insert, delete, update, reverse in the link list. |
| 15 | Write a program to create a linked list & perform operations such as insert, delete in the Circular link list. |
| 16 | Write a program, to Create a doubly linked list & perform search, insertion & delete operation |
|  |  |
|  |  |
| **F** | **TREE & GRAPH** |
| **17** | Write a program to implement binary search tree. (Insertion and Deletion in Binary Search Tree) |
| 18 | Write a program which simulates the various tree traversal algorithms. |
| 19 | Write program to simulate various graph traversing algorithms. |
|  |  |
| **G** | **SORTING ALGORITHMS** |
| 20 | Write program to implement Bubble, Insertion & selection sort |
| 21 | Write program to implement quick sort |
| 22 | Write program to implement merge sort. |
| 23 | Write a program to implement heap sort |

**A. ARRAY OPERATIONS**

1) Write a program to insert an element at given position in linear array.

#include <stdio.h>

int main()

{

int Array[10], Position, i, Number, Value;

printf("\n Enter number of elements in an array\n");

scanf("%d", &Number);

printf("\nPlease Enter %d elements \n", Number);

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

{

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

}

printf("\nPlease Enter the location where you wish to insert an element \n");

scanf("%d", &Position);

printf("\nPlease Enter the value to insert\n");

scanf("%d", &Value);

for (i = Number - 1; i>= Position - 1; i--)

{

Array[i+1] = Array[i];

}

Array[Position-1] = Value;

printf("\n Resultant array is\n");

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

{

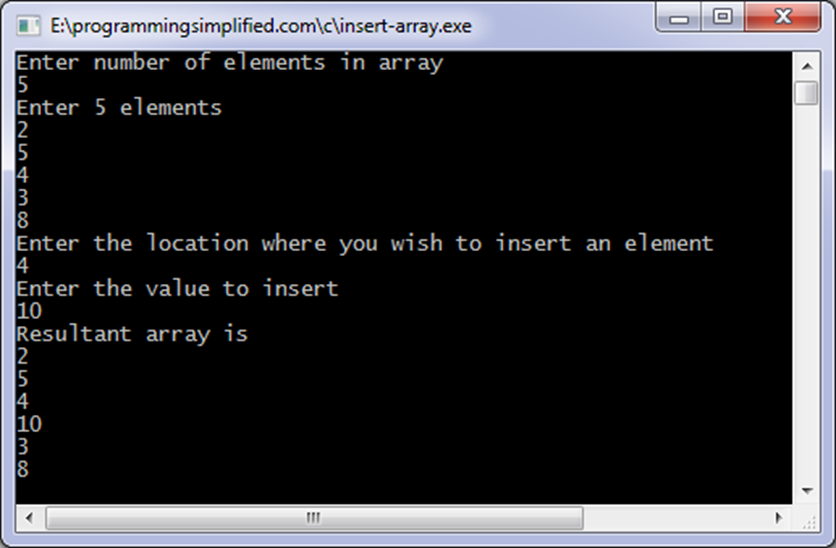
printf("%d\t", Array[i]);

}

return 0;

}

**Output:**



2) Write a program to insert an element in sorted array.

#include<stdio.h>

void main( )

{

int a[20],n,item,i;

printf("Enter the size of the array");

Scanf(“%d”,&n);

printf(" Array Elements in the sorted order\n");

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

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

}

printf("\nEnteritem to be inserted \n");

scanf("%d", &item);

i = n-1;

while(item<a[i] &&i>=0) {

a[i+1] = a[i];

i--;

}

a[i+1] = item;

n++;

printf("Sorted array after insertion:\n");

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

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

}

getch();

}

**Output:**

Enter the size of the array 5  
Array Elements in the sorted order 5   15   25   50   52  
Enter the item to be inserted 17  
Sorted array after insertion :  
5   15   **17**   25   50   52

3) Write a program to delete an element from given position in linear array.

#include <stdio.h>

int main()  
{  
   int array[100], position, c, n;

   printf("Enter number of elements in array**\n**");  
   scanf("%d", &n);

 printf("Enter %d elements**\n**", n);

   for (c = 0; c < n; c++)  
      scanf("%d", &array[c]);

   printf("Enter the location where you wish to delete element**\n**");  
   scanf("%d", &position);

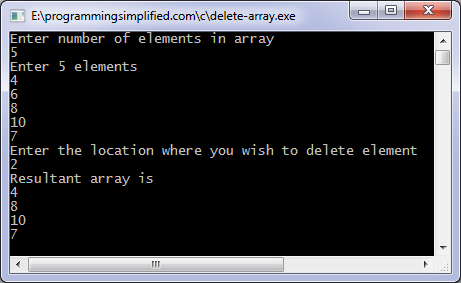
   if (position >= n+1)  
      printf("Deletion not possible.**\n**");  
   else  
   {  
      for (c = position - 1; c < n - 1; c++)  
         array[c] = array[c+1];

      printf("Resultant array:**\n**");

      for (c = 0; c < n - 1; c++)  
         printf("%d**\n**", array[c]);  
   }

   return 0;  
}

Output:



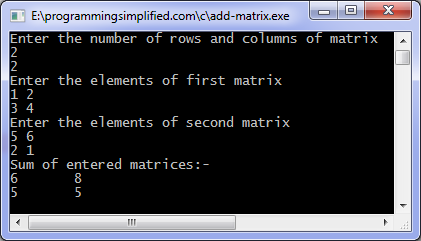
4.Perform following operations on matrices :

A) Addition B) Subtraction C) Multiplication D) Transpose

1. Addition:

#include <stdio.h>  
 int main()  
{  
   int m, n, c, d, first[10][10], second[10][10], sum[10][10];  
  printf("Enter the number of rows and columns of matrix**\n**");  
   scanf("%d%d", &m, &n);  
   printf("Enter the elements of first matrix**\n**");  
  for (c = 0; c < m; c++)  
      for (d = 0; d < n; d++)  
         scanf("%d", &first[c][d]);  
  printf("Enter the elements of second matrix**\n**");  
for (c = 0; c < m; c++)  
      for (d = 0 ; d < n; d++)  
         scanf("%d", &second[c][d]);  
  printf("Sum of entered matrices:-**\n**");  
   for (c = 0; c < m; c++) {  
      for (d = 0 ; d < n; d++) {  
         sum[c][d] = first[c][d] + second[c][d];  
         printf("%d**\t**", sum[c][d]);  
      }  
      printf("**\n**");  
    return 0;

Output:



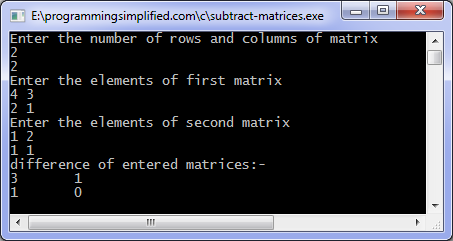
B) Subtraction:

#include <stdio.h>  
 int main()

{

int m, n, c, d, first[10][10], second[10][10], difference[10][10];  
  printf("Enter the number of rows and columns of matrix**\n**");  
   scanf("%d%d", &m, &n);  
   printf("Enter the elements of first matrix**\n**");  
  for (c = 0; c < m; c++)  
     for (d = 0 ; d < n; d++)  
       scanf("%d", &first[c][d]);  
 printf("Enter the elements of second matrix**\n**");  
 for (c = 0; c < m; c++)  
     for (d = 0; d < n; d++)  
  scanf("%d", &second[c][d]);  
 printf("Difference of entered matrices:-**\n**");  
for (c = 0; c < m; c++) {  
   for (d = 0; d < n; d++) {  
       difference[c][d] = first[c][d] - second[c][d];  
       printf("%d**\t**",difference[c][d]);  
     }  
     printf("**\n**");  
   }  
   
   return 0;  
}

Output:

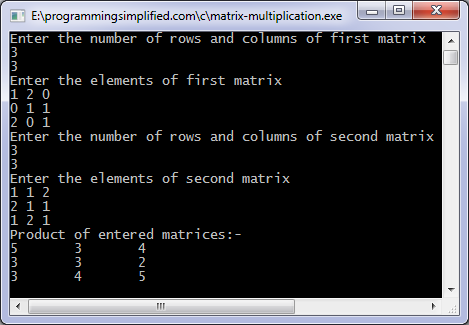


1. Multiplication:

#include <stdio.h>  
 int main()  
{  
  int m, n, p, q, c, d, k, sum = 0;  
  int first[10][10], second[10][10], multiply[10][10];  
  printf("Enter number of rows and columns of first matrix**\n**");  
  scanf("%d%d", &m, &n);  
  printf("Enter elements of first matrix**\n**");  
 for (c = 0; c < m; c++)  
    for (d = 0; d < n; d++)  
      scanf("%d", &first[c][d]);

printf("Enter number of rows and columns of second matrix**\n**");  
  scanf("%d%d", &p, &q);  
  if (n != p)  
    printf("The multiplication isn't possible.**\n**");  
  else  
  {  
    printf("Enter elements of second matrix**\n**");  
   
    for (c = 0; c < p; c++)  
      for (d = 0; d < q; d++)  
        scanf("%d", &second[c][d]);  
 for (c = 0; c < m; c++) {  
      for (d = 0; d < q; d++) {  
        for (k = 0; k < p; k++) {  
          sum = sum + first[c][k]\*second[k][d];  
        }  
 multiply[c][d] = sum;  
        sum = 0;  
      }  
    }  
   
    printf("Product of the matrices:**\n**");  
  for (c = 0; c < m; c++) {  
      for (d = 0; d < q; d++)  
   printf("%d**\t**", multiply[c][d]);  
  printf("**\n**");  
    }  
  }  
 return 0;  
}

Output:



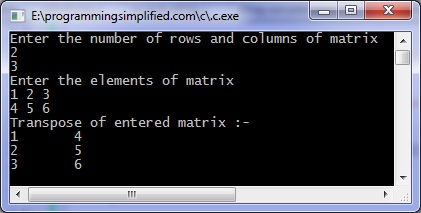
1. Transpose:

#include <stdio.h>  
 int main()  
{  
 int m, n, c, d, matrix[10][10], transpose[10][10];  
 printf("Enter the number of rows and columns of a matrix**\n**");  
  scanf("%d%d", &m, &n);

 printf("Enter elements of the matrix**\n**");  
  for (c = 0; c < m; c++)  
    for (d = 0; d < n; d++)  
      scanf("%d", &matrix[c][d]);  
  for (c = 0; c < m; c++)  
    for (d = 0; d < n; d++)  
      transpose[d][c] = matrix[c][d];  
 printf("Transpose of entered matrix:-**\n**");  
 for (c = 0; c < n; c++) {  
    for (d = 0; d < m; d++)  
      printf("%d**\t**", transpose[c][d]);  
    printf("**\n**");  
  }

  return 0;  
}

Output:



**B. SEARCHING**

5. Search an element in a linear array using linear search.

#include <stdio.h>

int main()  
{  
  int array[100], search, c, n;

printf("Enter number of elements in array**\n**");  
  scanf("%d", &n);

printf("Enter %d integer(s)**\n**", n);

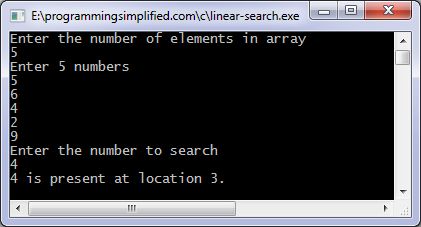
 for (c = 0; c < n; c++)  
    scanf("%d", &array[c]);

  printf("Enter a number to search**\n**");  
  scanf("%d", &search);

  for (c = 0; c < n; c++)  
  {  
    if (array[c] == search)    */\* If required element is found \*/*  
    {  
      printf("%d is present at location %d.**\n**", search, c+1);  
      **break**;  
    }  
  }  
  if (c == n)  
 printf("%d isn't present in the array.**\n**", search);

 return 0;  
}

Output:



**C. RECURSION**

7. Write a program to compute factorial of given number using recursion

#include<stdio.h>

long int multiplyNumbers(int n);

int main() {

int n;

printf("Enter the number : ");

scanf("%d",&n);

printf("The factorial of the number %d is

%ld", n, multiplyNumbers(n));

return 0;

}

long int multiplyNumbers(int n) {

if (n>=1)

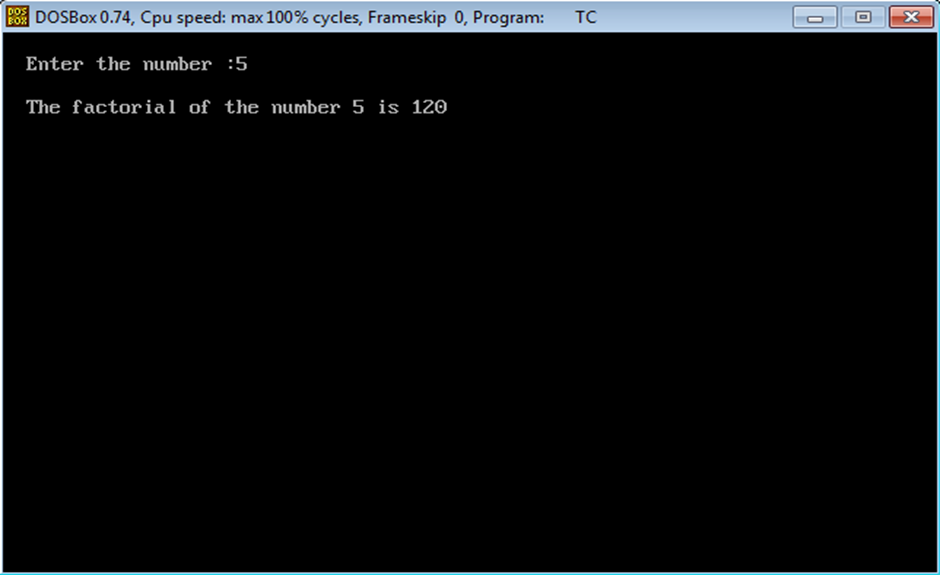
return n\*multiplyNumbers(n-1);

else

return 1;

}

Output:



8. Write a program to solve Tower of Hanoi problem using recursion.

#include<stdio.h>

void TOH(int,char,char,char);

void main()

{

int n;

printf("How many plates?");

scanf("%d",&n);

TOH(n,'A','B','C');

}

 void TOH(int n,charx,chary,char z)

{

if(n>0)

{

TOH(n-1,x,z,y);

printf("\n%c -> %c",x,y);

TOH(n-1,z,y,x);

}

}

Output:

*How many plates?3*

*A -> B  
A -> C  
B -> C  
A -> B  
C -> A  
C -> B  
A -> B*

9.Write a program to find power of given number using recursion.

#include <stdio.h>

double pow(double base, int expo);

int main()

{

double base, power;

int expo;

printf("Enter base: ");

scanf("%lf", &base);

printf("Enter exponent: ");

scanf("%d", &expo);

power = pow(base, expo);

printf("%.2lf ^ %d = %f", base, expo, power);

return 0;

double pow(double base, int expo)

{

if(expo == 0)

return 1;

else if(expo > 0)

return base \* pow(base, expo - 1);

else

return 1 / pow(base, -expo);

}

Output:put

Enter base: 2

Enter exponent: 5

2.00 ^ 5 = 32.000000

**D. STACK & QUEUE**

10.Write a program for static implementation of stack.

#include<stdio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

//clrscr();

top=-1;

printf("\n Enter the size of STACK[MAX=100]:");

scanf("%d",&n);

printf("\n\t STACK OPERATIONS USING ARRAY");

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

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

do

{

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

scanf("%d",&choice);

switch(choice) {

case 1: {

push();

break;

}

case 2: {

pop();

break;

}

case 3: {

display();

break;

}

case 4: {

printf("\n\t EXIT POINT ");

break;

}

default: {

printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

}

while(choice!=4);

return 0;

}

void push() {

if(top>=n-1) {

printf("\n\tSTACK is over flow");

}

else {

printf(" Enter a value to be pushed:");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

void pop() {

if(top<=-1)

{

printf("\n\t Stack is under flow");

}

else{

printf("\n\t The popped elements is %d",stack[top]);

top--;

}

}

void display() {

if(top>=0) {

printf("\n The elements in STACK \n");

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

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

printf("\n Press Next Choice");

}

else {

printf("\n The STACK is empty");

}

}

Output:

Enter the size of STACK[MAX=100]:10

STACK OPERATIONS USING ARRAY

--------------------------------

1.PUSH

2.POP

3.DISPLAY

4.EXIT

Enter the Choice:1

Enter a value to be pushed:12

Enter the Choice:1

Enter a value to be pushed:24

Enter the Choice:1

Enter a value to be pushed:98

Enter the Choice:3

The elements in STACK

98

24

12

Press Next Choice

Enter the Choice:2

The popped elements is 98

Enter the Choice:3

The elements in STACK

24

12

Press Next Choice

Enter the Choice:4

EXIT POINT

11. Write a program for dynamic implementation of queue.

#include <stdio.h>

#include<stdlib.h>

#define MAX 50

void insert();

void delete();

void display();

int queue\_array[MAX];

int rear = - 1;

int front = - 1;

int main() {

int choice;

while (1) {

printf("1.Insert element to queue n");

printf("2.Delete element from queue n");

printf("3.Display all elements of queue n");

printf("4.Quit n");

printf("Enter your choice : ");

scanf("%d", &choice);

switch(choice) {

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(1);

default:

printf("Wrong choice n");

}

}

}

void insert() {

int item;

if(rear == MAX - 1)

printf("Queue Overflow n");

else {

if(front== - 1)

front = 0;

printf("Inset the element in queue : ");

scanf("%d", &item);

rear = rear + 1;

queue\_array[rear] = item;

}

}

void delete() {

if(front == - 1 || front > rear) {

printf("Queue Underflow n");

return;

}

else

{

printf("Element deleted from queue is : %dn", queue\_array[front]);

front = front + 1;

}

}

void display() {

int i;

if(front == - 1)

printf("Queue is empty n");

else {

printf("Queue is : n");

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

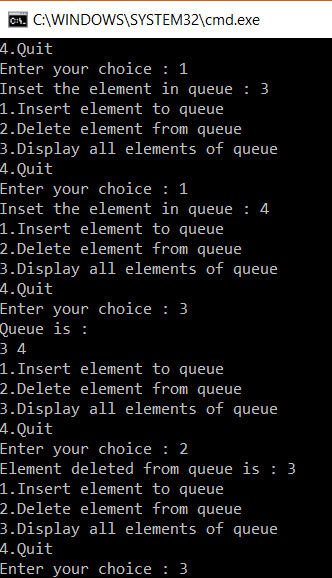
printf("%d ", queue\_array[i]);

printf("n");

}

}

Output:



12. Write a program for static implementation of circular queue.

#include<stdio.h>

# define MAX 5

int cqueue\_arr[MAX];

int front = -1;

int rear = -1;

void insert(int item)

{

if((front == 0 && rear == MAX-1) || (front == rear+1))

{

printf("Queue Overflow n");

return;

}

if(front == -1)

{

front = 0;

rear = 0;

}

else

{

if(rear == MAX-1)

rear = 0;

else

rear = rear+1;

}

cqueue\_arr[rear] = item ;

}

void deletion()

{

if(front == -1)

{

printf("Queue Underflown");

return ;

}

printf("Element deleted from queue is : %dn",cqueue\_arr[front]);

if(front == rear)

{

front = -1;

rear=-1;

}

else

{

if(front == MAX-1)

front = 0;

else

front = front+1;

}

}

void display()

{

int front\_pos = front,rear\_pos = rear;

if(front == -1)

{

printf("Queue is emptyn");

return;

}

printf("Queue elements :n");

if( front\_pos<= rear\_pos )

while(front\_pos<= rear\_pos)

{

printf("%d ",cqueue\_arr[front\_pos]);

front\_pos++;

}

Else {

while(front\_pos<= MAX-1) {

printf("%d ",cqueue\_arr[front\_pos])

front\_pos++;

}

front\_pos = 0;

while(front\_pos<= rear\_pos) {

printf("%d ",cqueue\_arr[front\_pos]);

front\_pos++;

}

}

printf("n");

}

int main()

{

int choice,item;

do {

printf("1.Insertn");

printf("2.Deleten");

printf("3.Displayn");

printf("4.Quitn");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice) {

case 1 :

printf("Input the element for insertion in queue : ");

scanf("%d", &item);

insert(item);

break;

case 2 :

deletion();

break;

case 3:

display();

break;

case 4:

break;

default:

printf("Wrong choicen");

}

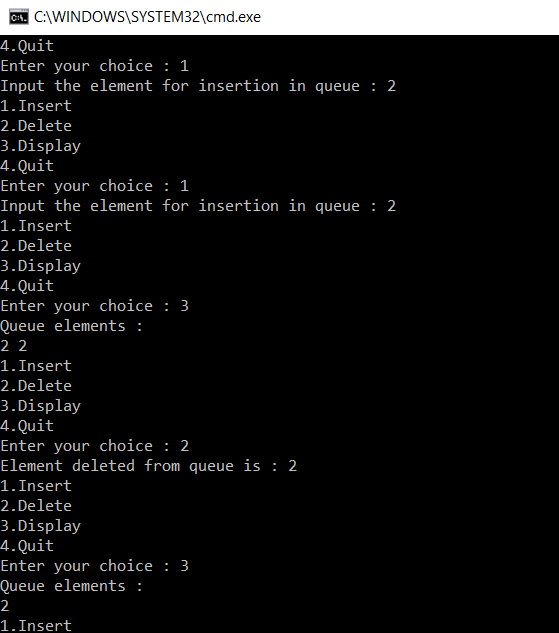
}

while(choice!=4);

return 0;

}

Output:



14. Write a program to evaluate a postfix operation.

#include<stdio.h>

int stack[20];

int top = -1;

void push(int x) {

stack[++top] = x;

}

int pop() {

return stack[top--];

}

int main() {

char exp[20];

char \*e;

int n1,n2,n3,num;

printf("Enter the expression : ");

scanf("%s",exp);

e = exp;

while(\*e != '\0') {

if(isdigit(\*e)) {

num = \*e - 48;

push(num);

}

else {

n1 = pop();

n2 = pop();

switch(\*e)

{

case '+': {

n3 = n1 + n2;

break;

}

case '-': {

n3 = n2 - n1;

break;

}

case '\*': {

n3 = n1 \* n2;

break;

}

case '/': {

n3 = n2 / n1;

break;

}

}

push(n3);

}

e++;

}

printf("\nThe result of expression %s is %d\n\n",exp,pop());

return 0;

}

## **OUTPUT:**

Enter the expression: 245+\*

The result of expression 245+\* is 18

**E. LINKED LIST**

15. Create a  linear linked list & perform operations such as  insert, delete at end , at beg & reverse the link list.

#include<stdlib.h>

#include <stdio.h>

void create();

void display();

void insert\_begin();

void insert\_end();

void insert\_pos();

void delete\_begin();

void delete\_end();

void delete\_pos();

struct node

{

        int info;

        struct node \*next;

};

struct node \*start=NULL;

int main()

{

        int choice;

        while(1){  printf("n                MENU                             n");

                printf("n 1.Create     n");

                printf("n 2.Display    n");

                printf("n 3.Insert at the beginning    n");

                printf("n 4.Insert at the end  n");

                printf("n 5.Insert at specified position       n");

                printf("n 6.Delete from beginning      n");

                printf("n 7.Delete from the end        n");

                printf("n 8.Delete from specified position     n");

                printf("n 9.Exit       n");

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

                printf("Enter your choice:t");

                scanf("%d",&choice);

                switch(choice)

                {

                        case 1:

                                        create();

                                        break;

                        case 2:

                                        display();

                                        break;

                        case 3:

                                        insert\_begin();

                                        break;

                        case 4:

                                        insert\_end();

                                        break;

                        case 5:

                                        insert\_pos();

                                        break;

                        case 6:

                                        delete\_begin();

                                        break;

                        case 7:

                                        delete\_end();

                                        break;

                        case 8:

                                        delete\_pos();

                                        break;

                        case 9:

                                        exit(0);

                                        break;

                        default:

                                        printf("n Wrong Choice:n");

                                        break;

                }

        }

        return 0;

}

void create()

{

        struct node \*temp,\*ptr;

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

        if(temp==NULL)

        {

                printf("nOut of Memory Space:n");

                exit(0);

        }

        printf("nEnter the data value for the node:t");

        scanf("%d",&temp->info);

        temp->next=NULL;

        if(start==NULL)

        {

                start=temp;

        }

        else

        {

                ptr=start;

                while(ptr->next!=NULL)

                {

                        ptr=ptr->next;

                }

                ptr->next=temp;

        }

}

void display()

{

        struct node \*ptr;

        if(start==NULL)

        {

                printf("nList is empty:n");

                return;

        }

        else

        {

                ptr=start;

                printf("nThe List elements are:n");

                while(ptr!=NULL)

                {

                        printf("%dt",ptr->info );

                        ptr=ptr->next ;

                }

        }

}

void insert\_begin()

{

        struct node \*temp;

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

        if(temp==NULL)

        {

                printf("nOut of Memory Space:n");

                return;

        }

        printf("nEnter the data value for the node:t" );

        scanf("%d",&temp->info);

        temp->next =NULL;

        if(start==NULL)

        {

                start=temp;

        }

        else

        {

                temp->next=start;

                start=temp;

        }

}

void insert\_end()

{

        struct node \*temp,\*ptr;

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

        if(temp==NULL)

        {

                printf("nOut of Memory Space:n");

                return;

        }

        printf("nEnter the data value for the node:t" );

        scanf("%d",&temp->info );

        temp->next =NULL;

        if(start==NULL)

        {

                start=temp;

        }

        else

        {

                ptr=start;

                while(ptr->next !=NULL)

                {

                        ptr=ptr->next ;

                }

                ptr->next =temp;

        }

}

void insert\_pos()

{

        struct node \*ptr,\*temp;

        int i,pos;

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

        if(temp==NULL)

        {

                printf("nOut of Memory Space:n");

                return;

        }

        printf("nEnter the position for the new node to be inserted:t");

        scanf("%d",&pos);

        printf("nEnter the data value of the node:t");

        scanf("%d",&temp->info) ;

        temp->next=NULL;

        if(pos==0)

        {

                temp->next=start;

                start=temp;

        }

        else

        {

                for(i=0,ptr=start;i<pos-1;i++) { ptr=ptr->next;

                        if(ptr==NULL)

                        {

                                printf("nPosition not found:[Handle with care]n");

                                return;

                        }

                }

                temp->next =ptr->next ;

                ptr->next=temp;

        }

}

void delete\_begin()

{

        struct node \*ptr;

        if(ptr==NULL)

        {

                printf("nList is Empty:n");

                return;

        }

        else

        {

                ptr=start;

                start=start->next ;

                printf("nThe deleted element is :%dt",ptr->info);

                free(ptr);

        }

}

void delete\_end()

{

        struct node \*temp,\*ptr;

        if(start==NULL)

        {

                printf("nList is Empty:");

                exit(0);

        }

        else if(start->next ==NULL)

        {

                ptr=start;

                start=NULL;

                printf("nThe deleted element is:%dt",ptr->info);

                free(ptr);

        }

        else

        {

                ptr=start;

                while(ptr->next!=NULL)

                {

                        temp=ptr;

                        ptr=ptr->next;

                }

                temp->next=NULL;

                printf("nThe deleted element is:%dt",ptr->info);

                free(ptr);

        }

}

void delete\_pos()

{

        int i,pos;

        struct node \*temp,\*ptr;

        if(start==NULL)

        {

                printf("nThe List is Empty:n");

                exit(0);

        }

        else

        {

                printf("nEnter the position of the node to be deleted:t");

                scanf("%d",&pos);

                if(pos==0)

                {

                        ptr=start;

                        start=start->next ;

                        printf("nThe deleted element is:%dt",ptr->info  );

                        free(ptr);

                }

                else

                {

                        ptr=start;

                        for(i=0;i<pos;i++) { temp=ptr; ptr=ptr->next ;

                                if(ptr==NULL)

                                {

                                        printf("nPosition not Found:n");

                                        return;

                                }

                        }

                        temp->next =ptr->next ;

                        printf("nThe deleted element is:%dt",ptr->info );

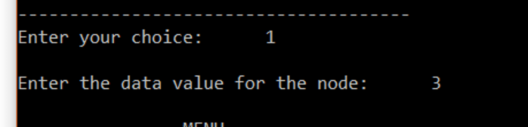
                        free(ptr);

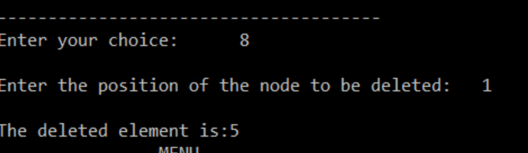
                }

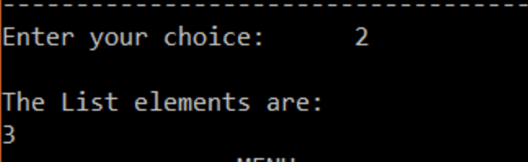
        }

}

Output:







16. Create a circular linked list & perform search, insertion & delete operation.

#include<stdio.h>

#include<stdlib.h>

 typedef struct Node {

int info;

struct Node \*next;

}

node;

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

 void create();

void del();

void display();

int main() {

int chc;

do {

printf("\nMenu\n\t 1 to create the element : ");

printf("\n\t 2 to delete the element : ");

printf("\n\t 3 to display the queue : ");

printf("\n\t 4 to exit from main : ");

printf("\nEnter your choice : ");

scanf("%d",&chc);

switch(chc) {

case 1:

create();

break;

case 2:

del();

break;

case 3:

display();

break;

case 4:

return 1;

default:

printf("\nInvalid choice :");

}

}while(1);

return 0;

}

void create() {

node \*newnode;

newnode=(node\*)malloc(sizeof(node));

printf("\nEnter the node value : ");

scanf("%d",&newnode->info);

newnode->next=NULL;

if(rear==NULL)

front=rear=newnode;

else {

rear->next=newnode;

rear=newnode;

}

rear->next=front;

}

void del() {

temp=front;

if(front==NULL)

printf("\nUnderflow :");

else {

if(front==rear) {

printf("\n%d",front->info);

front=rear=NULL;

}

else {

printf("\n%d",front->info);

front=front->next;

rear->next=front;

}

temp->next=NULL;

free(temp);

}

}

void display() {

temp=front;

if(front==NULL)

printf("\nEmpty");

else {

printf("\n");

for(;temp!=rear;temp=temp->next)

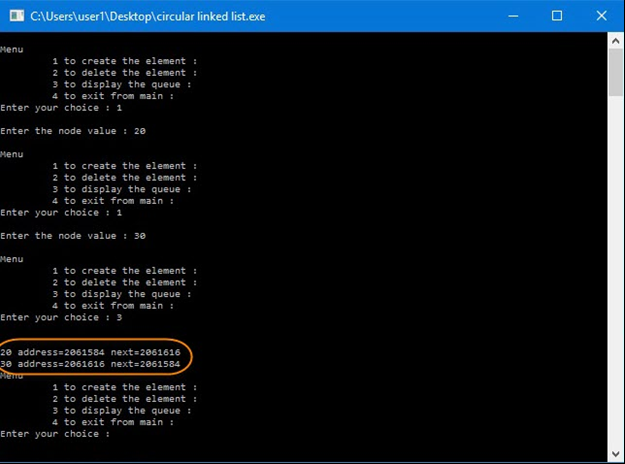
printf("\n%d address=%u next=%u\t",temp->info,temp,temp->next);

printf("\n%d address=%u next=%u\t",temp->info,temp,temp->next);

}

}

Output:



17. Create a doubly linked list &perform  search, insertion & delete operation.

#include<stdio.h>

#include<stdlib.h>

struct Node;

typedef struct Node \* PtrToNode;

typedef PtrToNode List;

typedef PtrToNode Position;

struct Node {

    int e;

    Position previous;

    Position next;

};

void Insert(int x, List l, Position p){

   Position TmpCell;

    TmpCell = (struct Node\*) malloc(sizeof(struct Node));

    if(TmpCell == NULL)

        printf("Memory out of spacen");

    else {

        TmpCell->e = x;

        TmpCell->previous = p;

        TmpCell->next = p->next;

        p->next = TmpCell;

    }

}

void Delete(int x, List l) {

    Position p, p1, p2;

    p = Find(x, l);

    if(p != NULL) {

        p1 = p -> previous;

        p2 = p -> next;

        p1 -> next = p -> next;

        if(p2 != NULL)                  // if the node is not the last node

            p2 -> previous = p -> previous;

    }

    else

        printf("Element does not exist!!!n");

}

void Display(List l) {

    printf("The list element are :: ");

    Position p = l->next;

    while(p != NULL)

    {

        printf("%d -> ", p->e);

        p = p->next;

    }

}

int main() {

    int x, pos, ch, i;

    List l, l1;

    l = (struct Node \*) malloc(sizeof(struct Node));

    l->previous = NULL;

    l->next = NULL;

    List p = l;

    printf("DOUBLY LINKED LIST IMPLEMENTATION OF LIST ADTnn");

    do {

        printf("nn1. CREATEn 2. DELETEn  3. DISPLAYn 4. QUITnnEnter the choice :: ");

        scanf("%d", &ch);

        switch(ch) {

        case 1:

            p = l;

            printf("Enter the element to be inserted :: ");

            scanf("%d",&x);

            printf("Enter the position of the element :: ");

            scanf("%d",&pos);

            for(i = 1; i<pos; i++) { p = p->next;

            }

            Insert(x,l,p);

            Break;

case 2:

            p = l;

            printf("Enter the element to be deleted :: ");

            scanf("%d",&x);

            Delete(x,p);

            break;

case 3:

            Display(l);

            break;

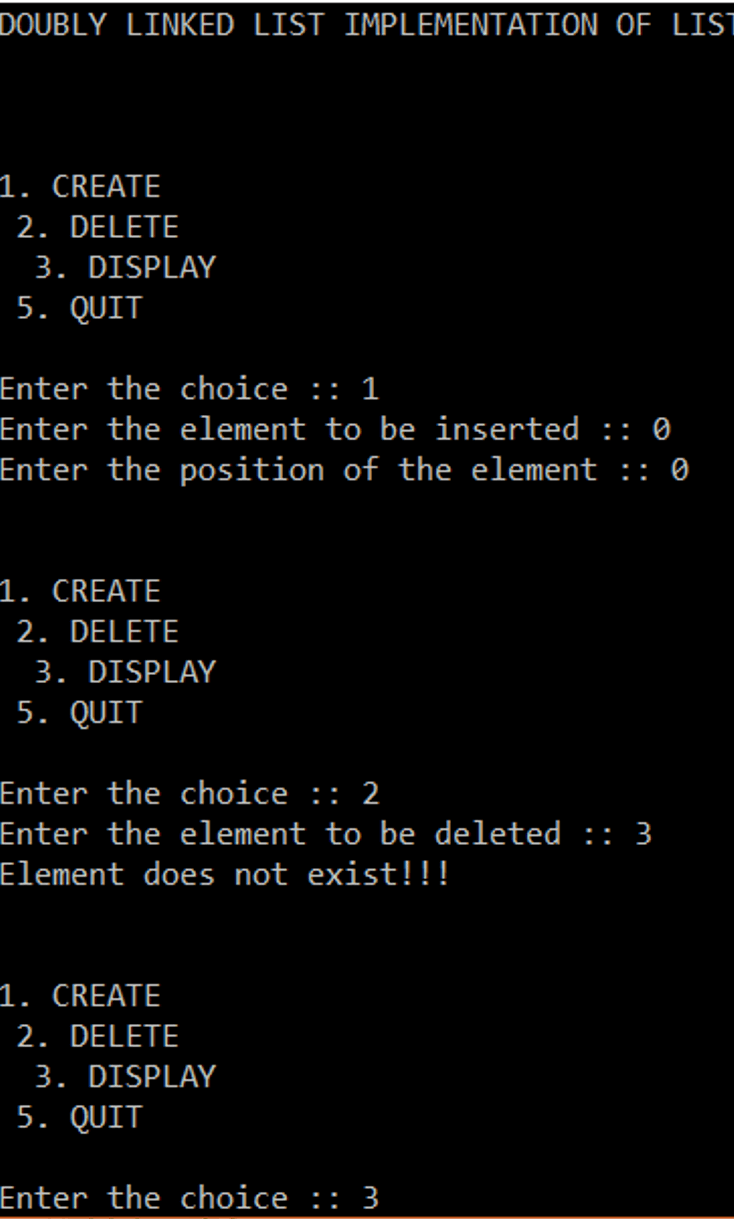
        }

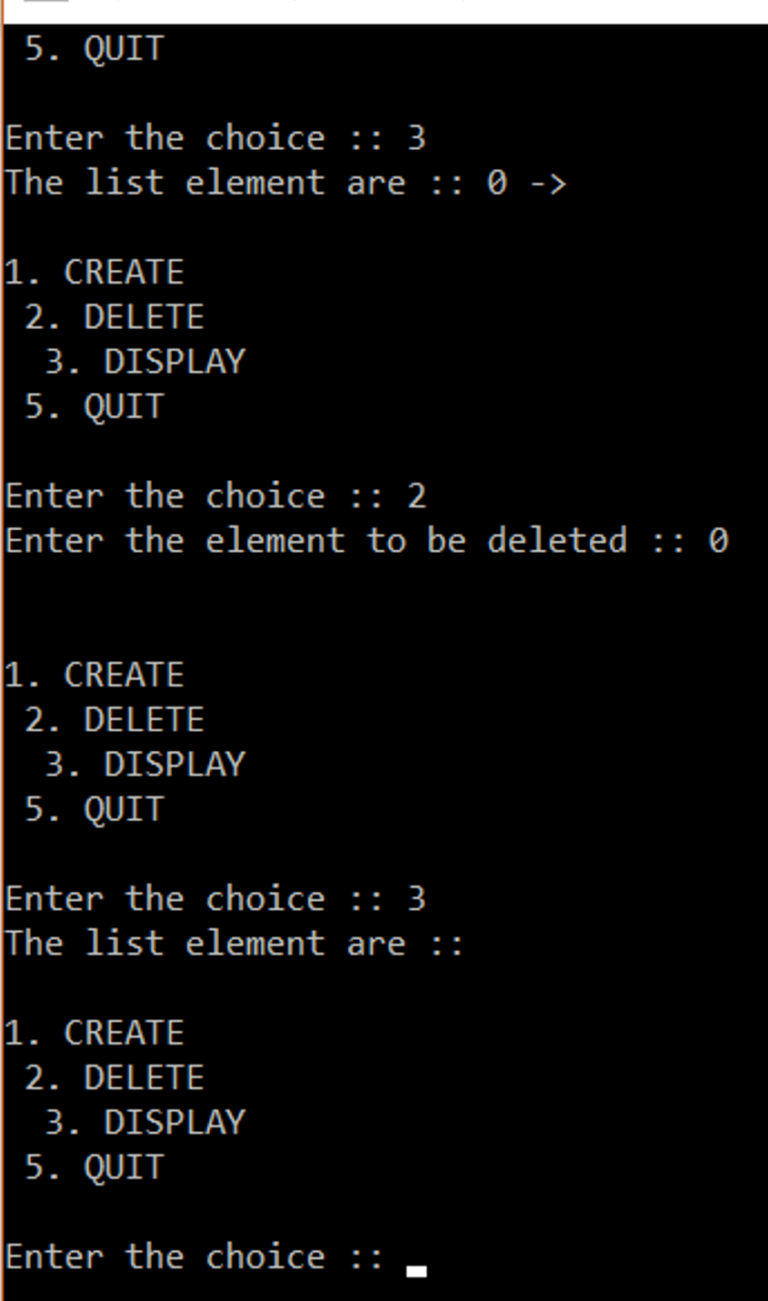
    }

    while(ch<4);

}

Output:





18. Write program to implement binary search tree. (Insertion and Deletion in Binary Search Tree).

#include<stdio.h>

#include<stdlib.h>

 typedef struct BST {

int data;

struct BST \*left;

struct BST \*right;

}node;

 node \*create();

void insert(node \*,node \*);

void preorder(node \*);

 int main(){

char ch;

node \*root=NULL,\*temp;

do {

temp=create();

if(root==NULL)

root=temp;

else

insert(root,temp);

printf("nDo you want to enter more(y/n)?");

getchar();

scanf("%c",&ch);

}while(ch=='y'|ch=='Y');

printf("nPreorder Traversal: ");

preorder(root);

return 0;

}

node \*create() {

node \*temp;

printf("nEnter data:");

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

scanf("%d",&temp->data);

temp->left=temp->right=NULL;

return temp;

}

void insert(node \*root,node \*temp) {

if(temp->data<root->data) {

if(root->left!=NULL)

insert(root->left,temp);

else

root->left=temp;

}

if(temp->data>root->data) {

if(root->right!=NULL)

insert(root->right,temp);

else

root->right=temp;

}

}

void preorder(node \*root) {

if(root!=NULL) {

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

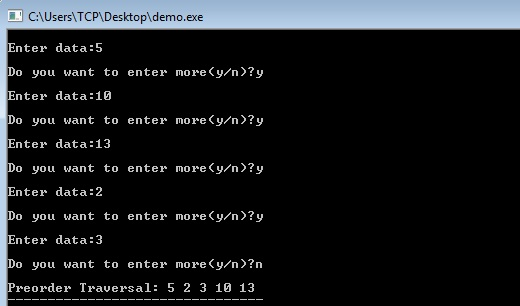
preorder(root->left);

preorder(root->right);

}

}

Output:



19. Write program to simulates the various tree traversal algorithms.

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*leftChild;

struct node \*rightChild;

};

struct node \*root = NULL;

void insert(int data) {

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

struct node \*current;

struct node \*parent;

tempNode->data = data;

tempNode->leftChild = NULL;

tempNode->rightChild =NULL;

if(root == NULL) {

root = tempNode;

} else {

current = root;

parent = NULL;

while(1) {

parent = current;

if(data < parent->data) {

current = current->leftChild;

if(current == NULL) {

parent->leftChild = tempNode;

return;

}

}

else {

current = current->rightChild;

if(current == NULL) {

parent->rightChild = tempNode;

return;

}

} } } }

struct node\* search(int data) {

struct node \*current = root;

printf("Visiting elements: ");

while(current->data != data) {

if(current != NULL)

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

if(current->data > data) {

current = current->leftChild;

}

else {

current = current->rightChild;

if(current == NULL) {

return NULL;

}

}

return current;

}

void **pre\_order\_traversal**(struct node\* root) {

if(root != NULL) {

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

pre\_order\_traversal(root->leftChild);

pre\_order\_traversal(root->rightChild);

}

}

void **inorder\_traversal**(struct node\* root) {

if(root != NULL) {

inorder\_traversal(root->leftChild);

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

inorder\_traversal(root->rightChild);

}

}

void **post\_order\_traversal**(struct node\* root) {

if(root != NULL) {

post\_order\_traversal(root->leftChild);

post\_order\_traversal(root->rightChild);

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

}

}

int main() {

int i;

int array[7] = { 27, 14, 35, 10, 19, 31, 42 };

for(i = 0; i< 7; i++)

insert(array[i]);

i = 31;

struct node \* temp = search(i);

if(temp != NULL) {

printf("[%d] Element found.", temp->data);

printf("\n");

}else {

printf("[ x ] Element not found (%d).\n", i);

}

i = 15;

temp = search(i);

if(temp != NULL) {

printf("[%d] Element found.", temp->data);

printf("\n");

}else {

printf("[ x ] Element not found (%d).\n", i);

}

printf("\nPreorder traversal: ");

pre\_order\_traversal(root);

printf("\nInorder traversal: ");

inorder\_traversal(root);

printf("\nPost order traversal: ");

post\_order\_traversal(root);

return 0;

}

Output:

Visiting elements: 27 35 [31] Element found.

Visiting elements: 27 14 19 [ x ] Element not found (15).

Preorder traversal: 27 14 10 19 35 31 42

Inorder traversal: 10 14 19 27 31 35 42

Post order traversal: 10 19 14 31 42 35 27

20. Write program to simulate various graph traversing algorithms.

Graph Traversal techinques:- Breadth First Search (BFS)

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #define MAX 100  #define initial 1  #define waiting 2  #define visited 3   int n;  int adj[MAX][MAX];  int state[MAX];  void create\_graph();  void BF\_Traversal();  void BFS(int v);   int queue[MAX], front = -1,rear = -1;  void insert\_queue(int vertex);  int delete\_queue();  int isEmpty\_queue();   int main()  {  create\_graph();  BF\_Traversal();  return 0;  }   void BF\_Traversal()  {  int v;  for(v=0; v<n; v++)  state[v] = initial;  printf("Enter Start Vertex for BFS: \n");  scanf("%d", &v);  BFS(v);  }   void BFS(int v)  {  int i;  insert\_queue(v);  state[v] = waiting;  while(!isEmpty\_queue())  {  v = delete\_queue( );  printf("%d ",v);  state[v] = visited;  for(i=0; i<n; i++)  {  if(adj[v][i] == 1 && state[i] == initial)  {  insert\_queue(i);  state[i] = waiting;  }  }  }  printf("\n");  }   void insert\_queue(int vertex)  {  if(rear == MAX-1)  printf("Queue Overflow\n");  else  {  if(front == -1)  front = 0;  rear = rear+1;  queue[rear] = vertex ;  }  }    int isEmpty\_queue()  {  if(front == -1 || front > rear)  return 1;  else  return 0;  }    int delete\_queue()  {  int delete\_item;  if(front == -1 || front > rear)  {  printf("Queue Underflow\n");  exit(1);  }    delete\_item = queue[front];  front = front+1;  return delete\_item;  }    void create\_graph()  {  int count,max\_edge,origin,destin;  printf("Enter number of vertices : ");  scanf("%d",&n);  max\_edge = n\*(n-1);    for(count=1; count<=max\_edge; count++)  {  printf("Enter edge %d( -1 -1 to quit ) : ",count);  scanf("%d %d",&origin,&destin);   if((origin == -1) && (destin == -1))  break;  if(origin>=n || destin>=n || origin<0 || destin<0)  {  printf("Invalid edge!\n");  count--;  }  else  {  adj[origin][destin] = 1;  }  }  }  Output:    Depth First Search (DFS) Adjacency Matrix : |

#include<stdio.h>

void DFS(int);

int G[10][10],visited[10],n;

void main()

{

int i,j;

printf("Enter number of vertices:");

scanf("%d",&n);

printf("\nEnteradjecency matrix of the graph:");

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

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

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

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

visited[i]=0;

DFS(0);

}

void DFS(int i)

{

int j;

printf("\n%d",i);

visited[i]=1;

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

if(!visited[j]&&G[i][j]==1)

DFS(j);

}

Output:

