Q2.Write a non recursive C program to implement inorder , preorder and postorder traversal for a BST.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 50
struct node
{
    struct node *left;
    int info;
    struct node *right;
};

struct node *insert(struct node *root, int ele );
void preorder(struct node *root);
void inorder(struct node *root);
void postorder(struct node *root);
void display(struct node *ptr,int level);
struct node *queue[MAX];
int front=-1,rear=-1;
void enqueue(struct node *item);
struct node *DeQue();
int Qempty();

struct node *stack[MAX];
int top=-1;
void Push(struct node *item);
struct node *Pop();
int StkEmt();

int main( )
```

```
{
    struct node *root=NULL;
    int x,y, ele;
    while(y=1)
    {
        printf("\n 1.Insert");
        printf("\n 2.Display");
        printf("\n 3.Preorder");
        printf("\n 4.Inorder");
        printf("\n 5.Postorder");
        printf("\n 6.Exit ");
        printf("\n Choose option:");
        scanf("%d",&x);
        switch(x)
        {

        case 1:
            printf("\nEnter Element : ");
            scanf("%d",&ele);
            root = insert(root, ele);
            break;

        case 2:
            printf("\n\t");
            display(root,0);
            printf("\n\t");
            break;

        case 3:
            preorder(root);
            break;
```

```
            case 4:
                inorder(root);
                break;

            case 5:
                postorder(root);
                break;

            case 6:
                exit(1);
                y=0;

            default:
                printf("\nWrong Input!!!x\n");
            }
        }
        return 0;
}



struct node *insert(struct node *root, int ele)
{
        struct node *tmp,*p,*ptr;
        ptr = root;
        p = NULL;
        while( ptr!=NULL)
        {
            p = ptr;
            if(ele < ptr->info)
                ptr = ptr->left;
```

```
            else
                ptr = ptr->right;
        }
        tmp=(struct node *)malloc(sizeof(struct node));
        tmp->info=ele;
        tmp->left=NULL;
        tmp->right=NULL;
        if(p==NULL)
            root=tmp;
        else if( ele < p->info )
            p->left=tmp;
        else
            p->right=tmp;
        return root;
}
void preorder(struct node *root)
{
        struct node *ptr = root;
        if( ptr==NULL )
        {
            printf("Tree is empty\n");
            return;
        }
        printf("\n Pre-order : ");
        Push(ptr);
        while( !StkEmt() )
        {
            ptr = Pop();
            printf("%d  ",ptr->info);
            if(ptr->right!=NULL)
                Push(ptr->right);
```

```
            if(ptr->left!=NULL)
                  Push(ptr->left);
      }
      printf("\t");
}
void inorder(struct node *root)
{
      struct node *ptr=root;

      if( ptr==NULL )
      {
            printf("Tree is empty\n");
            return;
      }
      printf("\n In-order : ");
      while(1)
      {
         while(ptr->left!=NULL )
            {
                  Push(ptr);
                  ptr = ptr->left;
            }
            while( ptr->right==NULL )
            {
                  printf("%d  ",ptr->info);
                  if(StkEmt())
                        return;
                  ptr = Pop();
            }
            printf("%d  ",ptr->info);
            ptr = ptr->right;
```

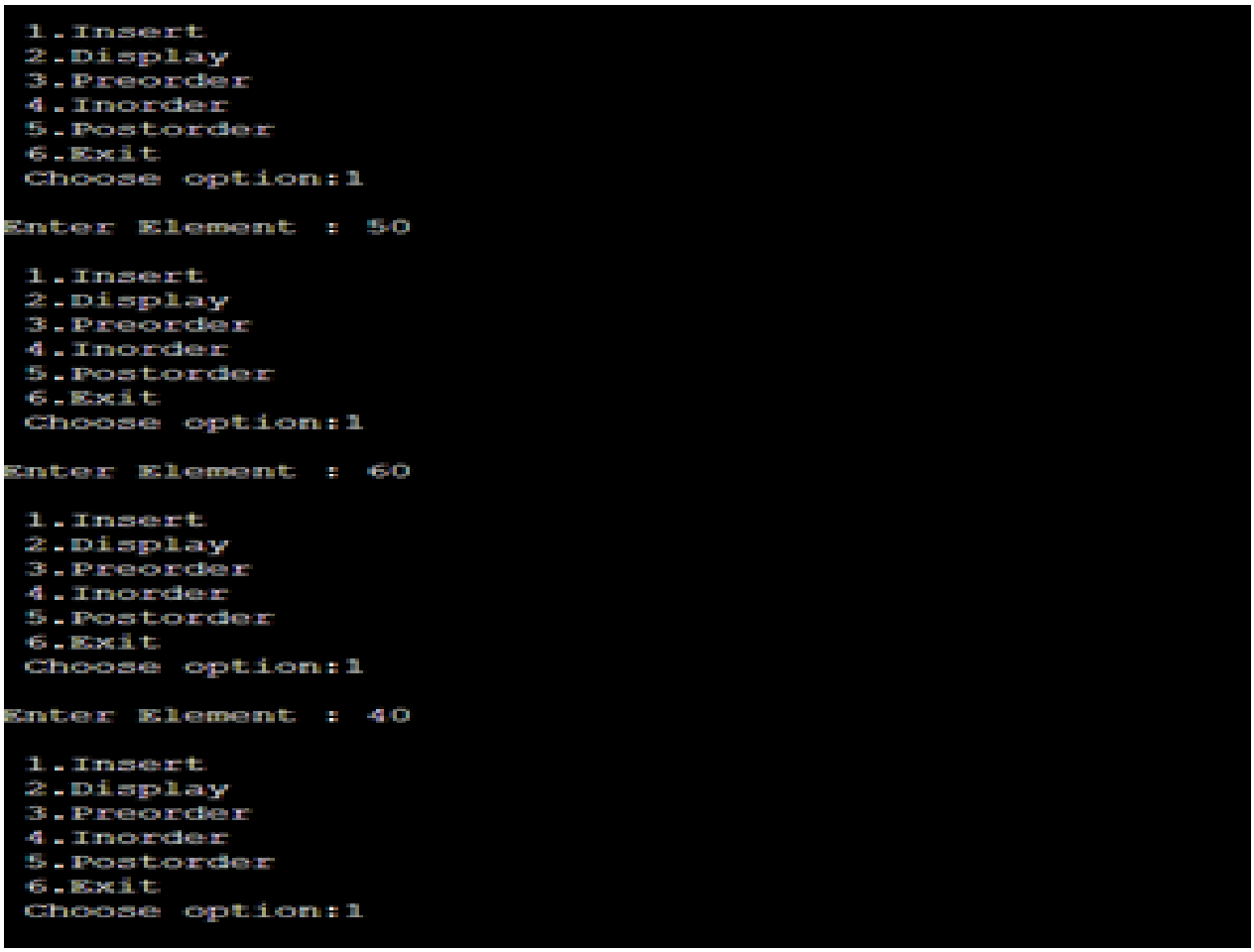
```
        }
}
void postorder(struct node *root)
{
        struct node *ptr = root;
        struct node *q;
        if( ptr==NULL )
        {
                printf("Tree is empty\n");
                return;
        }
        q = root;
        printf("\n Post-order : ");
        while(1)
        {
                while(ptr->left!=NULL)
                {
                        Push(ptr);
                        ptr=ptr->left;
                }
                while( ptr->right==NULL || ptr->right==q )
                {
                        printf("%d  ",ptr->info);
                        q = ptr;
                        if( StkEmt() )
                                return;
                        ptr = Pop();
                }
                Push(ptr);
                ptr = ptr->right;
        }
```

```
        printf("\t");
}
void enqueue(struct node *item)
{
        if(rear==MAX-1)
        {
                printf("queue Overflow\n");
                return;
        }
        if(front==-1)
                front=0;
        rear=rear+1;
        queue[rear]=item ;
}
struct node *DeQue()
{
        struct node *item;
        if(front==-1 || front==rear+1)
        {
                printf("queue Underflow\n");
                return 0;
        }
        item=queue[front];
        front=front+1;
        return item;
}
int Qempty()
{
        if(front==-1 || front==rear+1)
                return 1;
        else
```

```
        return 0;
}
void Push(struct node *item)
{
    if(top==(MAX-1))
    {
        printf("stack Overflow\n");
        return;
    }
    top=top+1;
    stack[top]=item;
}
struct node *Pop()
{
    struct node *item;
    if(top==-1)
    {
        printf("stack Underflow….\n");
        exit(1);
    }
    item=stack[top];
    top=top-1;
    return item;
}
int StkEmt()
{
    if(top==-1)
        return 1;
    else
        return 0;
}
```

```
void display(struct node *ptr,int level)
{
      int i;
      if(ptr == NULL )
            return;
      else
      {
         display(ptr->right, level+1);
         printf("\n\t");
         for (i=0; i<level; i++)
            printf("    ");
         printf("%d", ptr->info);
         display(ptr->left, level+1);
      }
}
```

OUTPUT:-



```
Enter Element : 35

 1.Insert
 2.Display
 3.Preorder
 4.Inorder
 5.Postorder
 6.Exit
 Choose option:1

Enter Element : 55

 1.Insert
 2.Display
 3.Preorder
 4.Inorder
 5.Postorder
 6.Exit
 Choose option:1

Enter Element : 70

 1.Insert
 2.Display
 3.Preorder
 4.Inorder
 5.Postorder
 6.Exit
 Choose option:2


                    70
               60
                    55
          50
               40
                    35
```

```
 1.Insert
 2.Display
 3.Preorder
 4.Inorder
 5.Postorder
 6.Exit
 Choose option:3

 Pre-order : 50  40  35  60  55  70
 1.Insert
 2.Display
 3.Preorder
 4.Inorder
 5.Postorder
 6.Exit
 Choose option:4

 In-order : 35  40  50  55  60  70
 1.Insert
 2.Display
 3.Preorder
 4.Inorder
 5.Postorder
 6.Exit
 Choose option:5

 Post-order : 35  40  55  70  60  50
 1.Insert
 2.Display
 3.Preorder
 4.Inorder
 5.Postorder
 6.Exit
 Choose option:6


...Program finished with exit code 0
Press ENTER to exit console.
```

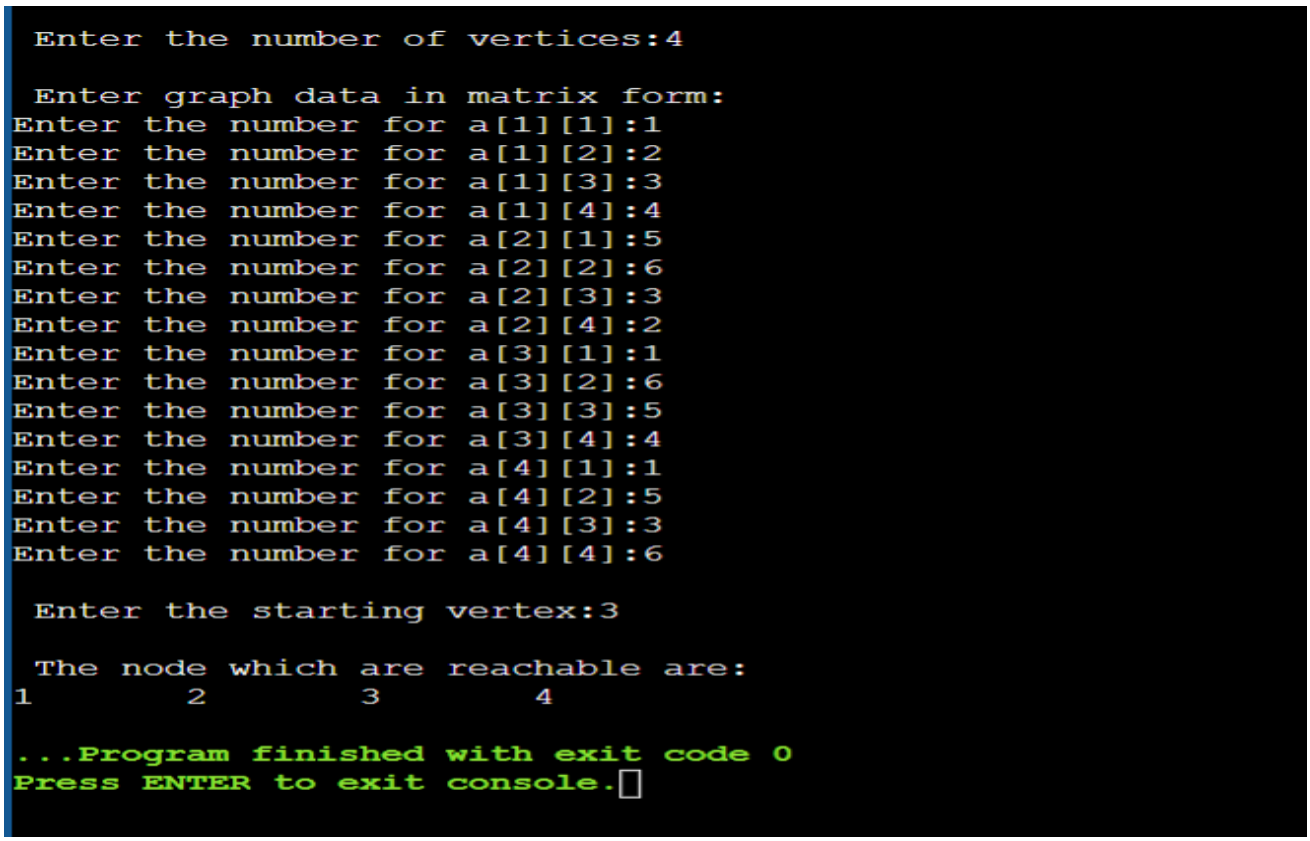
Q3.Write C program to implement Depth first search and Breadth first search traversals of a graph.

BFS PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v) {
 for(i = 1; i <= n; i++)
 if(a[v][i] && !visited[i])
 q[++r] = i;
 if(f <= r) {
 visited[q[f]] = 1;
 bfs(q[f++]);
 }
}

void main() {
 int v;
 printf("\n Enter the number of vertices:");
 scanf("%d", &n);

 for(i=1; i <= n; i++) {
 q[i] = 0;
 visited[i] = 0;
 }

 printf("\n Enter graph data in matrix form:\n");
 for(i=1; i<=n; i++) {
 for(j=1;j<=n;j++) {
printf("Enter the number for a[%d][%d]:",i,j);
 scanf("%d",&a[i][j]);
```

```
}
}

printf("\n Enter the starting vertex:");
scanf("%d", &v);
bfs(v);
printf("\n The node which are reachable are:\n");

for(i=1; i <= n; i++) {
if(visited[i])
printf("%d\t", i);
else {
printf("\n Bfs is not possible. Not all nodes are reachable");
break;
}
}
}
```

OUTPUT:



DFS PROGRAM:

```
include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
 int i;
 reach[v]=1;
 for(i=1;i<=n;i++)
  if(a[v][i] && !reach[i])
  {
   printf("\n %d->%d",v,i);
   dfs(i);
  }
}
void main()
{
 int i,j,count=0;
 printf("\n Enter number of vertices:");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 {
  reach[i]=0;
  for(j=1;j<=n;j++)
   a[i][j]=0;
 }
 printf("\n Enter the adjacency matrix:\n");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
   scanf("%d",&a[i][j]);
 dfs(1);
```

```
 printf("\n");
 for(i=1;i<=n;i++)
 {
  if(reach[i])
   count++;
 }
 if(count==n)
  printf("\n Graph is connected");
 else
  printf("\n Graph is not connected");
}
```

OUTPUT: