```c
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
struct node
{
    int data;
    struct node *left;
    struct node *right;
};
struct node *tree;
void create_tree(struct node *);
struct node *insert(struct node *,int val );
void preorder(struct node *);
void inorder(struct node *);
void postorder(struct node *);
struct node *smallest(struct node *tree);
void delete(struct node *tree,int data);
int main()
{
    int ch,val;
    create_tree(tree);
    do{
        printf("\n ****menu***");
        printf("\n 0:Exit");
        printf("\n 1. Insertion");
        printf("\n2.deletion:");
        printf("\n 3.preorder traversal");
        printf("\n 4.inorder traversal");
        printf("\n 5.postorder traversal");
        printf("\n Enter the choice =");
        scanf("%d",&ch);


        switch(ch)
        {
            case 1:printf("\nEnter the value=");
                   scanf("%d",&val);
                   tree=insert(tree,val);
                   break;



            case 3:
                  preorder(tree);
                  break;

            case 4:
                inorder(tree);
```

```
                break;

            case 5:
                postorder(tree);
                break;

            default:
                 printf("\n Invalid choice :");
                 break;
        }
    }
    while(ch!=0);
    return 0;
}
void create_tree(struct node *tree)
{
    tree=NULL;
}

struct node *insert(struct node *tree,int val)
{
    struct node *ptr,*nodeptr,*parentptr;
    ptr=(struct node*)malloc(sizeof(struct node));
    ptr->data=val;
    ptr->left=NULL;
    ptr->right=NULL;
    if(tree==NULL)
    {
        tree=ptr;
    }
    else
    {
        parentptr=NULL;
        nodeptr=tree;
        while(nodeptr!=NULL)
        {
            parentptr=nodeptr;
            if(val<nodeptr->data)
            {
                nodeptr=nodeptr->left;
            }
            else
            {
                nodeptr=nodeptr->right;
            }
        }
            if(val<parentptr->data)
            {
```

```
                ptr=parentptr->left;
            }
            else
            {
                ptr=parentptr->right;
            }


    }
    return tree;
}

void preorder(struct node *tree)
{
    if(tree!=NULL)
    {
    printf("\n \t %d",tree->data);
    preorder(tree->left);
    preorder(tree->right);
    }
}

void inorder(struct node *tree)
{
    if(tree!=NULL)
    {
        inorder(tree->left);
        printf("\t %d",tree->data);
        inorder(tree->right);
    }
}

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

void delete(struct node *tree,int data)
{
    if(tree==NULL)
    {
        tree=tree;
    }
```

```
    if(data>tree->data)
    {
        tree->right=delete(tree->right,data);
    }
    else if(data<tree->data)
    {
        tree->left=delete(tree->left,data);
    }
    else
    {
        //case 1:
        if(tree->left==NULL)
        {
            struct node *temp=tree->right;
            free(temp);
            return temp;
        }

        else if(tree->right==NULL)
        {
            struct node *temp=tree->left;
            free(temp);
            return temp;
        }

        else{
            struct node *temp=smallest(tree->right);
            tree->data=temp->data;
            tree->right=delete(tree->right,temp->data;);

        }
        return tree;
    }
}

struct node *smallest(struct node *tree)
{
  if(tree==NULL||tree->left==NULL)
  {
    return tree;
  }
  else{
    return smallest(tree->left);
  }
}
```

```c
#include<stdio.h>
#define max 10

int stack[max];
int top=-1;
void push(int );
void pop();
void peek();
void display();

int main()
{
    int ch,val;
    do
    {
        /* code */
        printf("\n *****menu*****");
        printf("\n 0.exit");
        printf("\n 1.PUSH");
        printf("\n 2.POP");
        printf("\n 3.PEEK");
        printf("\n 4.DISPLAY");

        printf("\n Enter the choice =");
        scanf("%d",&ch);



        switch (ch)
        {
        case /* constant-expression */1:
            /* code */push(val);
            break;

        case 2:
             pop();
             break;

        case 3:
             peek();
             break;

        case 4:
             display();
             break;
```

```
        default:
             printf("\n INVALID CHOICE ");
             break;
        }
    } while (ch!=0);
    return 0;
}

void push(int val)
{

    printf("\n enter the data you want to enter =");
    scanf("%d",&val);
    if(top==max-1)
    {
        printf("\n OVERFLOW");
    }

    else{
        top++;
        stack[top]=val;
    }
}

void pop()
{
    if(top==-1)
    {
        printf("\n UNDERFLOW");
    }
    else
    {
        top--;

    }
}

void peek()
{
    if(top==-1)
    {
        printf("\n UNDERFLOW");
    }
    else{
        printf("%d",stack[top]);
    }
}
```

```c
void display()
{
    int i;
    if(top==-1)
    {
        printf("\n UNDERFLOW");
    }
    else{
        for(i=0;i<=top;i++)
        {
            printf("\t %d",stack[i]);
        }
    }
}

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

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

struct node *top=0;

int push(int x);
int pop();
void peek();
void display();

int main()
{
    int val,i,x;
    int choice ;
    do
    {
      printf("\n ****menu*****");
      printf("\n 1.push");
      printf("\n 2.pop");
      printf("\n 3.peek");
      printf("\n 4.display");
      printf("\n 5. Exit");
      printf("\n Enter the choice =");
      scanf("%d",&choice);

      switch (choice)
      {
      case 1:
```

```
      printf("Enter the data you wanna insert=");
      scanf("%d",&val);
      push(x);
        /* code */
      break;

       case 2:
       pop();
       break;

       case 4:
       display();

      default:
      printf("\n Invalid choice ");
        break;
      }
        /* code */
    } while (choice!=5);

  return 0;
}

int push(int x)
{
    struct node *newnode;
    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=x;
    newnode->link=top;
    top=newnode;
}

void display()
{
    struct node *temp;
    temp=top;
    if(top==0)
    {
        printf("undeflow");
    }
    else
    {
        while (temp!=0)
        {
            printf("%d",temp->data);
            temp=temp->link;
        }
```

```
    }
}

void peek()
{
    if(top==NULL)
    {
        printf("undeflow");
    }
    else{
        printf("%d",top->data);
    }
}

int pop()
{
    struct node *temp;
    temp=top;
    if(top==0)
    {
      printf("underflow");
    }
    else
    {
        printf("%d",top->data);
        top=top->link;
        free(temp);
    }
}
```

```
#include<stdio.h>
#include<ctype.h>
#define max 100
float stack[max];
int top=-1;

void push(int);
float  pop();
float evaluatepostfix(char exp[]);
int main()
{
float val;
char exp[100];
```

```
printf("\n Enter the postfix expresion");
gets(exp);
val=evaluatepostfix(exp);
printf("\n the result=%.2f",val);
return 0;
}

void push(int val)
{

    if(top==max-1)
    {
        printf("\n OVERFLOW");
    }
    else{
        top++;
        stack[top]=val;
    }

}

float pop()
{

     return stack[top--];
}


float evaluatepostfix(char exp[])
{
    int i=0;
    float op1,op2,value;
    while(exp[i]!='\0')
    {
        if(isdigit(exp[i]))
        {
            push(exp[i]-'0');
        }
        else{
            op1=pop();
            op2=pop();
            switch (exp[i])
            {
            case /* constant-expression */'+':
                /* code */value=op1+op2;
                break;

            case  '-':
              value=op1-op2;
```

```
              break;

            case '*':
               value=op1*op2;
               break;

            case '/':
                value=op1/op2;
                 break;


            default:
                 printf("\n invalid choice");
                break;
            }
            push(value);
        }
        i++;

    }
    return (pop());
}
```

```
#include<stdio.h>
#include<ctype.h>
#define max 30
char stack[max];
int top=-1;

void push(char x);
int pop();
int priority(char x);

void push(char x)
{
    if(top==max-1)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        top++;
        stack[top]=x;
```

```
    }
}

int pop()
{
    if(top==-1)
    {
        printf("\n UNDERFLOW");
    }
    else{
    return stack[top--];
    }
}

int priority(char x)
{
    if(x=='(')
    {
        return 0;
    }

    if(x=='+'||x=='-')
    {
        return 1;
    }
    if(x=='*'||x=='/')
    {
       return 2;
    }
}

int main()
{
    char exp[100];
    char *e,x;
    printf("\n Enter the expression=");
    scanf("%s",&exp);
    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;

}
```

```
#include<stdio.h>
#define max 10
int q[max];
int f=-1,r=-1;

void insert(void);
int delete(void1);
void display(void);
int main()
{
  int ch,val,x;
  do{

    printf("\n******menu*******");
    printf("\n 1.insert");
    printf("\n 2.delete");
    printf("\n 3.display");
    printf("\n 4.exit");
```

```
    printf("\n Enter the choice ");
    scanf("%d",&ch);

    switch (ch)
    {
    case /* constant-expression */1:insert();
        /* code */
        break;

    case 2:
        val=delete();
        if(val!=-1)
        {
            printf("\n the number deleted is =%d",val);
        }
         break;




    case 3:display();
          break;


    default:
        printf("\n Invalid choice ");
        break;
    }
  }while(ch!=5);



}

void insert()
{
    int x;
    printf("\n Enter the number you want to insert=");
    scanf("%d",&x);

    if(f==0&&r==max-1)
    {
        printf("\n overflow");
    }
    else if(f==-1&&r==-1)
    {
        f=r=0;
        q[r]=x;
    }
```

```
    else if (r=max-1&&f!=0)
    {
        r=0;
        q[r]=x;

    }
    else{
       r++;
       q[r]=x;
    }
}


int  delete()
{
    int n;
    if(f==-1&&r==-1)
    {
        printf("\n UNDERFLOW");
    }
    n=q[f];
    if (f==r)
    {
        f=r=-1;
    }
    else{
        if(f==max-1)
        {
            f=0;
        }
        else{
            f++;
        }
    }
    return n;
}

void display()
{
    int i;
    if(f==-1&&r==-1)
    {
        printf("\nUNDERFLOW");
    }
    else
    {
        if(f<r)
        {
            for(i=f;i<=r;i++)
```

```
            {
              printf("\n %d",q[i]);
            }
        }
        else{
            for(i=f;i<max;i++)
            {
                printf("\n %d",q[i]);
            }
            for(i=0;i<=r;i++)
            {
                printf("\n %d",q[i]);
            }
        }

    }
}
```

```
#include<stdio.h>
#define max 10

int q[max];
int f=-1,r=-1;

void insert(void);
int delete(void);
void display(void);
int peek(void);

int main()
{
  int ch,val;
  do{

    printf("\n******menu*******");
    printf("\n 1.insert");
    printf("\n 2.delete");
    printf("\n 3.peek");
    printf("\n 4.display");
    printf("\n 5.exit");

    printf("\n Enter the choice ");
    scanf("%d",&ch);
```

```
    switch (ch)
    {
    case /* constant-expression */1:insert();
        /* code */
        break;

    case 2:

        val=delete();
        if(val!=-1)
        {
            printf("\n the number deleted is =%d",val);
        }
         break;


    case 3:peek();
        break;

    case 4:display();
          break;

    case 5:exit(0);
       break;

    default:
        printf("\n Invalid choice ");
        break;
    }
  }while(ch!=5);
    return 0;


}

void insert()
{
    int x;
    printf("\n Enter the data you want to insert ");
    scanf("%d",&x);
    if(r==max-1)
    {
        printf("overflow");
    }
    else if(f==-1&&r==-1)
    {
        f=r=0;
    }
    else{
```

```
        r++;
        q[r]=x;
    }
}

int delete()
{
    int n;
    if(f==-1||r==-1)
    {
        printf("underflow");
    }
    else{
        n=q[f];
        f++;
        if(f>r)
        {
            f=r=-1;

        }
        return n;

    }
}

void display()
{
    int i ;
    if(f==-1||f>r)
    {
        printf("\n queue is empty");
    }
    else{
        for(i=f;i<=r;i++)
        {
            printf("%d",q[i]);
        }
    }

}

int peek()
{
    if(f==-1||f>r)
    {
        printf("\n Queue is Empty");
    }
    else{
```

```
        return q[f];
        printf("%d",q[f]);
    }
}
```

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
struct node {
    int data;
    struct node *next;
};

struct node *front=NULL;
struct node *rear=NULL;
void enqueue(int val);
void dequeue();
void display();
int main()
{
   int ch,x;
   do
   {
    /* code */
    printf("\n **********menu*********");
    printf("\n 1.enqueue");
    printf("\n 2.dequeue");
    printf("\n 3.display");
    printf("\n Enter the choice =");
    scanf("%d",&ch);
    switch (ch)
    {
    case /* constant-expression */1:
                printf("\n enter the value you want to insert =");
                scanf("%d",&x);
                enqueue(x);
                break;

    case 2:
         dequeue();
         break;
```

```
    case 3:
        display();
        break;

    default:
        break;
    }
   } while (ch!=0);

}

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

void dequeue()
{
    struct node *temp;
    temp=front;
    if(front==NULL&&rear==NULL)
    {
        printf("\n UNDERFLOW");
    }
    else{
        front=front->next;
        free(temp);
    }
}

void display()
{
    struct node *temp;
    if(front==NULL&&rear==NULL)
    {
        printf("\n UNDERFLOW");
    }
```

```
    else{
        temp=front;
        while(temp!=NULL)
        {
            printf("\t  %d",temp->data);
            temp=temp->next;
        }
    }
}
```

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
struct node
{
    /* data */
    int data;
    struct node *next;
};

struct node *start=NULL;
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *insert_before(struct node *);
struct node *insert_after(struct node *);
struct node *delete(struct node *);
struct node *delete_end(struct node *);
struct node *display(struct node *start);

int main()
{
   int ch;
  do
  {
    /* code */
    printf("\n **********MENU**********");
    printf("\n0.exit");
    printf("\n1.insert from beginning");
    printf("\n2.insert from end");
    printf("\n3.insert from before the node");
```

```
    printf("\n4.insert from after the node");
    printf("\n5.delete from beginning");
    printf("\n6.delete from the end");
    printf("\n7.display linked list ");
    printf("\n Enter the choice =");
    scanf("%d",&ch);

    switch (ch)
    {
    case /* constant-expression */1:
        /* code */start=insert_beg(start);
        break;

    case 2:
       start=insert_end(start);
       break;

    case 3:
        start=insert_beg(start);
        break;

    case 4:
       start=insert_after(start);
       break;

     case 5:
         start=delete(start);
         break;

    case 6:
       start=delete_end(start);
       break;

    case 7:
        start=display(start);
        break;
    default:
    printf("\n Invalid choice ");
        break;
    }
  } while (ch!=0);


}
struct node *insert_beg(struct node *start)
{
    int num;
    struct node *newnode;
```

```
    printf("\n Enter the data=");
    scanf("%d",&num);
    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=num;
    newnode->next=start;
    start=newnode;
    return start;
}

struct node *insert_end(struct node *start)
{
    int num;
    struct node *newnode ,*ptr;
    printf("\n Enter the data =");
    scanf("%d",&num);
    newnode=(struct node *)malloc(sizeof(struct node*));
    newnode->data=num;
    newnode->next=NULL;
    ptr=start;
    while(ptr->next!=NULL)
    {
        ptr=ptr->next;
    }
    ptr->next=newnode;
    return start;
}

struct node *insert_before(struct node *start)
{
    int num,val;
    struct node *preptr,*ptr,*newnode;
    printf("\n Enter the data=");
    scanf("%d",&num);
    printf("\n Enter the value you want to add before ");
    scanf("%d",&val);
    newnode=(struct node *)malloc(sizeof(struct node));
    newnode->data=num;
    ptr=start;
    while(ptr->data!=val)
    {
        preptr=ptr;
        ptr=ptr->next;
    }
    preptr->next=newnode;
    newnode->next=ptr;
}

struct node *insert_after(struct node *start)
```

```
{
    int num,val;
    struct node *preptr,*ptr,*newnode;
    printf("\n enter the data =");
    scanf("%d",&num);
    printf("\n Enter the data you want ot add before ");
    scanf("%d",&val);
    newnode=(struct node *)malloc(sizeof(struct node));
    newnode->data=num;
    ptr=start;
    preptr=ptr;
    while(ptr->data!=val)
    {
        preptr=ptr;
        ptr=ptr->next;
    }
    preptr->next=newnode;
    newnode->next=ptr;
}

struct node *delete(struct node *start)
{
    struct node *ptr;
    ptr=start;
    start=start->next;
    free(ptr);
    return start;
}

struct node *delete_end(struct node *start)
{
    struct node *ptr,*preptr;
    ptr=start;
    while(ptr->next!=NULL)
    {
        preptr=ptr;
        ptr=ptr->next;
    }
    preptr->next=NULL;
    free(ptr);
    return start;
}

struct node *display(struct node *start)
{
    struct node *ptr;
    ptr=start;
    while(ptr!=NULL)
```

```
    {
        printf("\t %d",ptr->data);
        ptr=ptr->next;
    }
    return start;
}
```

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

struct queue
{
    int size;
    int f;
    int r;
    int *arr;
};


int isEmpty(struct queue *q){
    if(q->r==q->f){
        return 1;
    }
    return 0;
}

int isFull(struct queue *q){
    if(q->r==q->size-1){
        return 1;
    }
    return 0;
}

void enqueue(struct queue *q, int val){
    if(isFull(q)){
        printf("This Queue is full\n");
    }
    else{
        q->r++;
        q->arr[q->r] = val;
        // printf("Enqued element: %d\n", val);
    }
}

int dequeue(struct queue *q){
```

```
    int a = -1;
    if(isEmpty(q)){
        printf("This Queue is empty\n");
    }
    else{
        q->f++;
        a = q->arr[q->f];
    }
    return a;
}

int main(){
    // Initializing Queue (Array Implementation)
    struct queue q;
    q.size = 400;
    q.f = q.r = 0;
    q.arr = (int*) malloc(q.size*sizeof(int));

    // BFS Implementation
    int node;
    int i = 1;
    int j;
    int visited[5]={0,0,0,0,0};
    int a[5][5]={
         {0,1,0,1,0},
         {1,0,1,0,0},
         {0,1,0,1,1},
         {1,0,1,0,1},
         {0,0,1,1,0}

    };
    printf("%d", i);
    visited[i] = 1;
    enqueue(&q, i); // Enqueue i for exploration
    while (!isEmpty(&q))
    {
        int node = dequeue(&q);
        for(j=1; j<5 ;j++)
        {
            if(a[node][j] ==1 && visited[j] == 0){
                printf("%d", j);
                visited[j] = 1;
                enqueue(&q, j);
            }
        }
    }
    return 0;
}
```

```c
#include <stdio.h>

int a[20][20], reach[20], n;

void dfs(int v)
{
    int i;
    reach[v] = i;
    for (i = 1; i <= n; i++)
    {
        if (a[v][i] && !reach[i])
        {
            printf("\n %d->%d", v, i);
            dfs(i);
        }
    }
}

int main()
{
    int i, j, count = 0;
    printf("\n Enter the 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);


    return 0;
}
```