10

```
#include<stdio.h>
#include<conio.h>
int q[20],ele,i,f=0,r=-1,c=0,qs=8,k;
//Here f is front end,r is rear end and c is count
//qs is Queue Size
//To insert element
void Insert()
{
 if(c==qs)
 printf("Queue Overflow");
 else
 { r=(r+1)%qs;
 printf("Enter element");
 scanf("%d",&ele);
 q[r]=ele;
 c++;
 }
}
//To Delete element
void Delete()
{
 if(c==0)
```

```
 printf("Queue Underflow");
 else
 {
 printf("%d has been deleted from queue",q[f]);
 f=(f+1)%qs;
 c--;
 }
}
//To Display elements
void Display()
{
 if (c==0)
 printf("Queue Underflow");
 else
 {
 for(i=1,k=f;i<=c;i++)
 {
 printf("%d\n",q[k]);
 k=(k+1)%qs;
 }
 }
}
void main()
{
```

```
 int choice,z;
 While(i)
{
 printf("\nMENU\n1=Insert\n2=Delete\n3=Display\n4=EXIT\nEnter your choice");
 scanf("%d",&z);
 switch(z)
 {
 case 1:Insert();
 break;
 case 2:Delete();
 break;
 case 3:Display();
 break;
 case 4: exit(0);
 }
}
 getch();
}
```

11

```
#include <stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *link;
};
struct node *first = NULL;
void add_front()
{
 struct node *temp;
 temp = (struct node *)malloc(sizeof(struct node));
 printf("\nEnter data : ");
 scanf("%d",&temp->data);

 if(first == NULL)
 {
 first = temp;
 first->link=NULL;
 }
 else
 {
```

```
 temp->link = first;
 first = temp;
 }
 printf("\nNode added(f)");
}
void del_front()
{
 struct node *temp;

 if(first == NULL)
 printf("\nThe list is empty");
 else
 {
 temp = first;
 first = first->link;
 free(temp);
 printf("\nNode deleted(f)");
 }
}
void add_rear()
{
 struct node *temp,*ptr;
 temp = (struct node *)malloc(sizeof(struct node));
 printf("\nEnter data : ");
```

```
 scanf("%d",&temp->data);
 temp->link = NULL;

 if(first == NULL)
 {
 first = temp;
 printf("\nNode added(r)");
 }
 else
 {
 ptr = first;
 while(ptr->link!=NULL)
 {
 ptr = ptr->link;
 }
 ptr->link = temp;
 printf("\nNode added(r)");
 }
}
void del_rear()
{
 struct node *cur, *prev;

 if(first == NULL)
```

```
 printf("\nThe list is empty");


 else if(first->link==NULL)
 {
 free(first);
 first=NULL;
 printf("\nNode deleted(r)");
 }
 else
 {
 cur = first;
 while(cur->link!=NULL)
 {
 prev = cur;
 cur = cur->link;
 }
 free(cur);
 prev->link = NULL;
 printf("\nNode deleted(r)");
 }
}
void display()
{
 struct node *ptr;
```

```
 if(first == NULL)
 printf("\nThe list is empty");
 else
 {
 ptr = first;
 printf("\nThe element(s) are :");
 while(ptr!=NULL)
 {
 printf("\n%d",ptr->data);
 ptr=ptr->link;
 }
 }
}
int main()
{
 int choice,c;
 do{
 printf("\nEnter Choice :\n1-Add Front\n2-Delete Front\n3-Add
Rear\n4-Delete Rear\n5-Display");
 printf("\nAnswer :");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1 : add_front();
```

```
 break;
 case 2 : del_front();
 break;
 case 3 : add_rear();
 break;
 case 4 : del_rear();
 break;
 case 5 : display();
 break;
 default : printf("\nIncorrect Choice");
 }
 printf("\n\n\tpress '1' to continue :");
 scanf("%d",&c);
 }while(c==1);
 return 0;
}
```

12

```
#include<stdio.h>
#include<conio.h>
struct node
{
 int co,po;
 struct node *link;
};
struct node *f1=NULL,*f2=NULL,*f3=NULL;

struct node* create(struct node *first)
{ int i=1;
  struct node *tmp;
   struct node *cur;
 printf("Enter cooefecient and Power in descending order of the polynimioal\n");
 while(i)
 {
  tmp=(struct node*)malloc(sizeof(struct node));
  tmp->link=NULL;
  printf("enter coeff and power");
  scanf("%d%d",&tmp->co,&tmp->po);
```

```
 if(first==NULL)
  first=tmp;
 else
 {
  cur=first;
  while(cur->link!=NULL)
  {
   cur=cur->link;
  }
  cur->link=tmp;
 }
 printf("1-> continue , 0-> exit");
 scanf("%d", &i);
 }
 return first;
}

void display(struct node *first)
{
  struct node *cur;
  if(first==NULL)
  {
    printf("NO LINKED LIST");
  }
```

```
  else
  {
    cur=first;
    while(cur!=NULL)
    {
          if(cur->co >0)
          {
            printf("+%dx%d  ",cur->co,cur->po);
          }
          else
          {
            printf("%dx%d  ",cur->co,cur->po);
          }
          cur=cur->link;
    }
   }
}

void final( int co ,int po)
 {
   struct node *tmp , * cur;
   tmp=(struct node*)malloc(sizeof(struct node));
   tmp->link=NULL;
   tmp->po=po ;
```

```
    tmp->co=co;
   if(f3==NULL)
    f3=tmp;
   else
   {
    cur=f3;
    while(cur->link!=NULL)
    {
     cur=cur->link;
    }
    cur->link=tmp;
   }
}

 void process(struct node *f1 ,  struct node *f2)
  {
     struct node *a , *b;
     a=f1;
     b=f2;
     while(a!=NULL && b!=NULL)
     {
          if(a->po == b->po)
           {
            final(a->co+b->co, a->po);
```

```
   a=a->link;
   b=b->link;
  }
  else if( a->po > b->po)
   {
    final(a->co,a->po);
    a=a->link;
    }
  else
  {
    final(b->co,b->po);
    b=b->link;
  }
}
while(a!=NULL)
{
  final(a->co,a->po);
    a=a->link;
}
while(b!=NULL)
{
  final(b->co,b->po);
    b=b->link;
}
```

```
 }



void main()
{
printf("Input the first poly\n");
f1=create(f1);
printf("input the 2nd poly\n");
f2=create(f2);
process(f1,f2);
printf("1st poly\n");
display(f1);
printf("\n 2nd poly\n");
display(f2);
printf(" \nthe sum of two poly\n")   ;
display(f3);
}
```

13

```
//Double Linked List


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


struct node
{
  int data;
  struct node *llink,*rlink;
} ;


struct node *first=NULL;


void add_front()
{
  struct node *temp;
  temp=(struct node*)malloc(sizeof(struct node));
  printf("\nInput the data:");
  scanf("%d",&temp->data);
  if(first==NULL)
  {
    first=temp;
```

```
   first->llink=NULL;
   first->rlink=NULL;
  }
  else
  {
   temp->llink=NULL;
   temp->rlink=first;
   first->llink=temp;
   first=temp;
  }
}

void Delete_front()
{
  struct node *temp;
  if(first==NULL)
   printf("\nLinked List is empty");
  else if(first->rlink==NULL)
  {
   printf("\n%d is deleted",first->data);
   free(first);
   first=NULL;
  }
  else
```

```
  {
    printf("\n%d is deleted",first->data);

    temp=first;
    first=first->rlink;
    first->llink=NULL;
    free(temp);
  }

}

void add_rear()
{
  struct node *temp;
  temp=(struct node*)malloc(sizeof(struct node));
  printf("\nInput the data:");
  scanf("%d",&temp->data);
  if(first==NULL)
  {
    first=temp;
    first->llink=NULL;
    first->rlink=NULL;
  }
  else
```

```
  {
    struct node *current;
    current=first;
    while(current->rlink!=NULL)
      current=current->rlink;
    current->rlink=temp;
    temp->llink=current;
    temp->rlink=NULL;
  }
}

void Delete_rear()
{
  if(first==NULL)
    printf("\nLinked List is empty");
  else if(first->rlink==NULL)
  {
    printf("\n%d is deleted",first->data);
    free(first);
    first=NULL;
  }
  else
  {
    struct node *temp;
```

```
   struct node *current;
   current=first;
   while(current->rlink!=NULL)
     current=current->rlink;
   temp=current->llink;
   printf("\n%d is deleted",current->data);
   free(current);
   temp->rlink=NULL;
  }

}


void add_specific()
{
  struct node *temp,*current;
  int key; // key stores the element agter which new element will be added
  temp=(struct node*)malloc(sizeof(struct node));
  printf("\nInput the data:");
  scanf("%d",&temp->data);
  printf("\nEnter the element after which new element should be added:");
  scanf("%d",&key);
  current=first;
  while(1)
```

```
{
  if(current->data==key)
  {
    if(current->rlink==NULL)
    {
      current->rlink=temp;
      temp->llink=current;
      temp->rlink=NULL;
      return;
    }
    else
    {
      temp->rlink=current->rlink;
      current->rlink=temp;
      temp->llink=current;
      current=temp->rlink;
      current->llink=temp;
      return;
    }
  }
  if(current->rlink==NULL)
  {
    printf("\n%d not found",key);
    return;
```

```
    }
  current=current->rlink;
  }
}

void Delete_specific()
{
  int key; //key stores the element to be deleted
  struct node *current,*previous;
  printf("\nInput the data to be deleted:");
  scanf("%d",&key);
  if(first==NULL)
  {
   printf("\nLinked List is empty");
   return;
  }
  current=first;
  while(1)
  {
    if(current->data==key)
    {
      if(current->llink==NULL&&current->rlink==NULL)
      {
        printf("\n%d is deleted",current->data);
```

```
  free(current);
  first=NULL;
  return;
}
else if(current->llink==NULL)
{
  printf("\n%d is deleted",current->data);
  previous=current;
  current=current->rlink;
  current->llink=NULL;
  first=current;
  return;
}
else if(current->rlink==NULL)
{
  printf("\n%d is deleted",current->data);
  previous=current->llink;
  previous->rlink=NULL;
  free(current);
  return;
}
else
{
  previous=current->llink;
```

```
        previous->rlink=current->rlink;
        printf("\n%d is deleted",current->data);
        free(current);
        current=previous->rlink;
        current->llink=previous;
        return;
       }

    }
    if(current->rlink==NULL)
    {
      printf("\n%d not found",key);
      return;
    }
  current=current->rlink;
  }
}


void display()
{
  struct node *current;
  current=first;
  if(first==NULL)
```

```
   printf("\nLinked List is empty");
  else
  {
   while(current!=NULL)
   {
    printf("\n%d",current->data);
    current=current->rlink;
   }
  }

}

int main()
{
  int choice;
  while(1)
  {
   printf("\n\nEnter your choice");
   printf("\n1. Add Front\t2. Delete Front\n3. Add Rear\t4. Delete Rear\n5. Add Specific\t6.
Delete Specific\n7. Display\t8. Exit\nCHOICE:");
   scanf("%d",&choice);
   switch (choice)
   {
    case 1:
```

```
  add_front();
  break;

case 2:
  Delete_front();
  break;

case 3:
  add_rear();
  break;

case 4:
  Delete_rear();
  break;

case 5:
  add_specific();
  break;

case 6:
  Delete_specific();
  break;

case 7:
```

```
        display();
        break;

      case 8:
        exit(0);
    }
  }
  return 0;
}
```

14

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

struct node
{
  struct node *llink;
  int data;
  struct node * rlink;
} ;

struct node *root=NULL;




void create()
{
  struct node * cur, *prev,*temp;
  int n ,ele,i;
  printf("enter the no of nodes\n");
  scanf("%d",&n);
```

```
for (i=0;i<n;i++)
{
   temp=(struct node*)malloc(sizeof(struct node));
   printf("\nInput the data:");
   scanf("%d",&temp->data);
        temp->rlink=NULL;
        temp->llink=NULL;
    if(root==NULL)
     {
      root=temp;
     }
    else
     {
      cur=root;

      while(cur!=NULL)
       {
                prev=cur;
                if(temp->data < cur->data)
                        {
                          cur=cur->llink;
                        }
              else
                        {
```

```
                                cur=cur->rlink;
                               }
          }

     if( temp->data < prev->data)
          prev->llink=temp;
     else
          prev->rlink=temp;
        }
     }
}

void preorder(struct node *r)
 {
   if(r!=NULL)
          {
                    printf(" %d" , r->data);
                    preorder(r->llink);
                    preorder(r->rlink);
          }
 }

 void inorder(struct node *r)
 {
```

```
   if(r!=NULL)
          {      inorder(r->llink);
                    printf(" %d" , r->data);

                    inorder(r->rlink);
          }
  }

  void postorder(struct node *r)
  {
    if(r!=NULL)
          {

                    postorder(r->llink);
                    postorder(r->rlink);
                    printf(" %d" , r->data);
          }
  }
void main()

{
printf("tree creation\n");
create();
printf("preorder traversal\n");
```

```
preorder(root);
printf("inorder\n");
inorder(root);
printf("postorder\n");
postorder(root);
getch();
}
```

15

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

struct node
{
   struct node *llink;
   int data;
   struct node *rlink;
};

struct node *root = NULL;

void insert()
{
   struct node *temp, *cur, *prev;

   temp = (struct node*)malloc(sizeof(struct node));
   printf("\nEnter Value : ");
   scanf("%d",&temp->data);
   temp->llink = NULL;
   temp->rlink = NULL;
```

```
if(root == NULL)
root = temp;

else
{
   cur = root;
   prev = NULL;

   while(cur!=NULL)
   {
      prev = cur;
      if(temp->data < cur->data)
         cur = cur->llink;
      else if(temp->data > cur->data)
         cur = cur->rlink;
      else
      {
         printf("\n\tDuplicate not allowed");
         return;
      }
   }

   if(temp->data < prev->data)
      prev->llink = temp;
```

```
        else
            prev->rlink = temp;
    }
}
```

```
void delete()
{
    int ele,flag=0;
    struct node *temp, *cur, *prev, *suc, *q;

    if(root==NULL)
    {
        printf("\n\tTree is Empty");
        return;
    }

    printf("\nEnter Value to be Deleted: ");
    scanf("%d",&ele);
```

```
cur = root;
prev = NULL;

while(cur!=NULL)
{

   if(cur->data == ele)
   {
      flag = 1;
      break;
   }
   else if(ele < cur->data)
   {
      prev = cur;
      cur = cur->llink;
   }
   else
   {
      prev = cur;
      cur = cur->rlink;
   }

}
```

```
if(flag==0)
{
   printf("\n\tElement not found");
}
else
{
   if (cur->llink == NULL)
      q = cur->rlink;
   else if (cur->rlink == NULL)
      q = cur->llink;
   else
   {
      suc = cur->rlink;

      while(suc->llink!=NULL)
      {
         suc = suc->llink;
      }

      suc->llink = cur->llink;
      q = cur->rlink;
   }

   if(cur == root)
```

```
        {
            free(root);
            root = q;
            return;
        }

        if(cur == prev->llink)
        {
            prev->llink = q;
            free(cur);
        }
        else
        {
            prev->rlink = q;
            free(cur);
        }
    }

}

void display(struct node *r)
{
    if(r!=NULL)
    {
```

```
        display(r->llink);
        printf("%d\t",r->data);
        display(r->rlink);
    }
}


int main()
{
    int ch;
    while(1)
    {
        printf("\nEnter Choice\n1-Insertion\n2-Deletion\n3-Display\n4-Exit\n");
        printf("\nAnswer :");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1 : insert();
                     break;
            case 2 : delete();
                     break;
            case 3 : if(root == NULL)
                       printf("\n\tTree is Empty");
```

```
  else
                {
                  printf("\nElements of the Tree(In-Order) :");
                  display(root);
                }
                break;
          case 4 : exit(0);

          default : printf("\n\tInvalid Choice");
       }
    }
    return 0;
}
```

16

BFS

```
#include<stdio.h> #include<conio.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]==0)
        q[++r]=i;
      }
   if(f<=r)
    {
      visited[q[f]]=1;
      bfs(q[f++]);
    }
}
void main()
 {
           int v; clrscr();
```

```
        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++)
                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");
        getch();
}
```

dfs

```
#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;
clrscr();
printf("\n Enter number of vertices:");
scanf("%d",&n);
        for(i=1;i<=n;i++)
```

```
        {
        reach[i]=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");

}
```