DSC LAB

1. SPARSE MATRIX

```
#include<stdio.h>

typedef struct
{
int row;
int col;
int val;
}term;

void createTriplet(term t[],int a[][10],int m, int n)
{

int i,j,k=0;
t[0].row=m;
t[0].col=n;
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
 if(a[i][j]==0)
    continue;
 k++;
 t[k].row=i;
 t[k].col=j;
```

```
 t[k].val=a[i][j];
 }
}
 t[0].val=k;
}

void printTriplet(term t[],int k)
{
int i;
printf("Triplet representation of sparse matrix\n");
for(i=0;i<k;i++)
{
 printf("%d\t",t[i].row);
 printf("%d\t",t[i].col);
 printf("%d\t",t[i].val);
 }
}
void findTranspose(term t1[], term t2[])
{
   int i,j,k;
   t2[0].row=t1[0].col;
   t2[0].col=t1[0].row;
   t2[0].val=t1[0].val;
   k=1;
   for(i=0;i<t1[0].col;i++)
   {
```

```
    for(i=0;i<t1[0].col;i++)
    {
     if(t1[j].col==i)
        {
          t2[k].row=t1[j].col;
          t2[k].col=t1[j].row;
          t2[k].val=t1[j].val;
          k++;
        }
     }
    }
  }

int main()
{
     int a[10][10],i,j,m,n,zero=0,nonzero;
    term t1[101],t2[102];
    printf("enter the dimension of the matrix");
    scanf("%d%d",&m,&n);
    printf("enter the elements");
    for(i=0;i<m;i++)
    {
      for(j=0;j<n;j++)
      {
         scanf("%d",&a[i][j]);
         if(a[i][j]==0)
```

```
            zero++;
        }
    }
    if(zero>(m*n)/2)
    {
        nonzero=m*n-zero;
        printf("the matrix has %d zeros\n",zero);
        printf(" It is a sparse matrix\n");
        printf(" Sparse matrix representation of given matrix is\n");
        createTriplet(t1,a,m,n);
        printTriplet(t1,nonzero+1);
        findTranspose(t1,t2);
        printf("Transpose of sparse matrix is\n");
        printTriplet(t2,nonzero+1);
    }
    else
    {
        printf("the matrix has %d zeros",zero);
        printf("The given matrix is not a sparse matrix\n");
    }
}
```

2. INSERTION SORT

```
#include<stdio.h>
void create(int a[],int n)
{
int i;
for(i=0;i<n;i++)
{
scanf("%d", &a[i]);
}
}
void display(int a[], int n)
{
int i;
for(i=0;i<n;i++)
{
printf("%d", a[i]);
}
}
void insort(int a[], int n)
{
int i,j,num;
for(i=0;i<n;i++)
{
num=a[i];
j=i-1;
while(j>=0 && num<a[j])
```

```
{
a[j+1]=a[j];
j=j-1;
}
a[j+1]=num;
}
}
int main()
{
int a[10];
int n;
printf("Enter the value of n: ");
scanf("%d", &n);
printf("Enter the terms \n");
create(a,n);
printf("\nArray before the sort: \n");
display(a,n);
insort(a,n);
printf("\nArray after sort: \n");
display(a,n);
return(0);
}
```

3. SHELL SORT

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

void createArray(int a[], int n)
{
  int i;
  for(i=0;i<n;i++)
     scanf("%d",&a[i]);
}

void display(int a[], int n)
{
   int i;
   for (i=0;i<n;i++)
      printf("%d\t",a[i]);
}

void shellSort(int a[], int n)
{
  int i,j,k,gap,temp;

    for ( gap = n/2; gap>=1; gap=gap/2)
    {
       for (j=gap;j<n;j++)
       {
```

```
          for (k=j-gap; k>=0;k=k-gap)
          {
             if(a[k+gap]>a[k])
               break;
             else
             {
               temp=a[k+gap];
               a[k+gap]=a[k];
               a[k]=temp;
             }
          }
        }
     }
}

void main()
{
   int n,a[10];
   printf("Enter the number of elements:");
   scanf("%d",&n);
   printf("Enter %d elements\n",n);
   createArray(a,n);
   printf("Array before sorting\n");
   display(a,n);
   shellSort(a,n);
   printf("\nArray after sorting\n");
```

```
    display(a,n);
}
```

4. QUICK SORT

```
#include<stdio.h>
void create(int a[], int n)
{
int i;
for(i=0;i<n;i++)
{
scanf("%d", &a[i]);
}
}
void display(int a[], int n)
{
int i;
for(i=0;i<n;i++)
{
printf("%d\n", a[i]);
}
}
int part(int a[], int low, int high)
{
int piv,i,j,temp;
piv=a[high];
i=low-1;
for(j=low;j<=high-1;j++)
{
if(a[j]<piv)
```

```
{
i++;
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
temp=a[i+1];
a[i+1]= a[high];
a[high]=temp;
return(i+1);
}
void quicksort(int a[], int low, int high)
{
int pos;
if(low<high)
{
pos=part(a,low,high);
quicksort(a,low,pos-1);
quicksort(a,pos+1,high);
}
}
int main()
{
int a[10];
int n;
```

```
printf("Enter the value of n: ");
scanf("%d", &n);
printf("\nEnter the terms: \n");
create(a,n);
printf("\nArray before sort: \n");
display(a,n);
quicksort(a,0,n-1);
printf("\nArray after sort: \n");
display(a,n);
return(0);
}
```

5. TOH AND ACKERMANN

T O H

```
#include<stdio.h>
void towerHanoi(int n,char from,char to,char aux)
{
   if(n==1)
   {
      printf("\n move disk 1 from peg %c to peg %c",from,to);
      return;
   }
   towerHanoi(n-1,from,aux,to);
   printf("\n move disk %d from peg %c to peg %c",n,from,to);
   towerHanoi(n-1,aux,to,from);
}
int main()
{
   int num;
   printf("enter the number of disks: ");
   scanf("%d",&num);
   printf("the sequence of moves in the %d disk tower of Hanoi are:
\n",num);
   towerHanoi(num,'A','C','B');
}
```

ACKERMANN

```
#include<stdio.h>
int ackerman(int m, int n)
{
   if(m==0)
   {
      return n+1;
   }
   else if ((m>0)&&(n==0))
   {
```

```
        return ackerman(m-1,1);
    }
    else if ((m>0)&&(n>0))
    {
        return ackerman(m-1,ackerman(m,n-1));
    }
}
int main()
{
    int m,n,result;
    printf("enter the value of M and N\n");
    scanf("%d%d",&m,&n);
    result=ackerman(m,n);
    printf("A(%d,%d)=%d\n",m,n,result);
}
```

6. STACK

```
#include<stdio.h>
#define n 5
int stack[n],top=-1;

void push(int x)
{
   if(top>=n-1)
   {
      printf("\nSTACK OVERFLOW");
   }
   else
   {
      top++;
      stack[top]=x;
   }
}
int pop()
{
   int x;
   if(top<=-1)
   {
      printf("\nSTACK UNDERFLOW");
      return 0;
   }
   else
   {
      x=stack[top];
      top--;
      return x;
   }

 }
void display()
{
   int i;
```

```
    if(top<0)
    {
       printf("\nStack is EMPTY");
    }
    else
    {
      printf("\nThe elements in the stack are:");
      for(i=top;i>=0;i--)
       printf("\n%d",stack[i]);
    }
}
void main()
{
    int choice,x,item;
    printf("Imlementation of stack using array\n");
    printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT");
    do
    {
       printf("\nEnter the choice:");
       scanf("%d",&choice);
       switch(choice)
          {
             case 1:
               printf("\n Enter the value to be pushed:");
               scanf("%d",&x);
               push(x);
               break;

             case 2:
               item=pop();
               if(item!=0)
               printf("The popped element is:%d",item);
               break;

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

```
            case 4:
              printf("\nEXIT");
              break;

            default:
              printf("\n Enter a valid choice(1 to 4)");

          }
        }
        while(choice!=4);

}
```

7. INFIX TO POSTFIX

```
#include<stdio.h>
#include<ctype.h>
#define SIZE 50
// we did a global delcaration
char stack[SIZE];
int top=-1;
void push(char x)
{
   top++;
   stack[top]=x;
   // stack [++top]=x;
}
char pop()
{
   char x;
   x=stack[top];
   top--;
   return x;
}

int pri(char symbol)
{
   if(symbol=='^')
   {
      return(6);
   }
   else if (symbol=='*'|| symbol=='/')
   {
      return(3);
   }
   else if (symbol=='+'|| symbol=='-')
   {
      return(1);
   }
   else
```

```
    {
        return(0);
    }

}
int prs(char symbol)
{
    if(symbol=='^')
    {
        return(5);
    }
    else if (symbol=='*'|| symbol =='/')
    {
        return(4);
    }
    else if (symbol=='+'|| symbol =='-')
    {
        return(2);
    }
    else
    {
        return(0);
    }

}
int main()
{
    char infix[50],postfix[50],ch,elem;
    int i=0,k=0;
    printf("Enter infix Expresion: ");
    scanf("%s",infix);
    push('#');
    while((ch=infix[i])!='\0')
    {
        if(ch=='(')
        push(ch);
        else if (isalnum(ch))
```

```
        {
           postfix[k++]=ch;
        }
        else if (ch==')')
        {
           while(stack[top]!='(')
            postfix[k++]=pop();
           elem=pop();
        }
        else
        {
           while(pri(ch)<=prs(stack[top]))
           postfix[k++]=pop();
           push(ch);
        }
     i++;
    }
    while(stack[top]!='#')
    postfix[k++]=pop();
    postfix[k]='\0';
    printf("\n postfix expression = %s\n",postfix);
}
```

8. EVALUATION OF POSTFIX

```
#include<stdio.h>
#include<ctype.h>
char postfix[20];
float stack[20];
int top=-1;
void push(float ch)
{
   top++;
   stack[top]=ch;
}

float pop()
{
   float x;
   x=stack[top];
   top--;
   return x;
}

void main()
{
   float op1,op2;
   int i;
   printf("Enter postfix expression:\n");
   scanf("%s",postfix);
   for(i=0;postfix[i]!='\0';i++)
   {
      if(isdigit(postfix[i]))
      {
         push(postfix[i]-48);
      }
      else
      {
         op2=pop();
         op1=pop();
```

```
        switch(postfix[i])
        {
           case '+':
                 push(op1+op2);
                 break;
           case '-':
                 push(op1-op2);
                 break;
           case '*':
                 push(op1*op2);
                 break;
           case '/':
                 if(op2==0)
                    printf("Divide by zero\n");
                 else
                 {
                    push(op1/op2);
                 }
                 break;
           default:
                 printf("Wrong Operator\n");
                 break;

        }
      }

   }
   printf("Result is %f\n",stack[top]);
}
```

9. QUEUE

```
#include<stdio.h>
#define n 5
int queue[n],front=-1,rear=-1;
void enqueue(int x)
{
 if(rear==n-1)
 {
 printf("\nQUEUE OVERFLOW\n");
 }
 else
 {
 if(front==-1)
 {
 front=0;
 printf("\n%d",front);
 }
 rear++;
 printf("\n%d",rear);
 queue[rear]=x;
 }
}
int dequeue()
{
 int y;
 if(front==-1)
 {
 printf("\nUNDERFLOW\n");
 return 0;
 }
 else if(front==rear)
 {
 y=queue[front];
 front=rear=-1;
 }
 else
```

```
 {
 y=queue[front];
 front++;
 }
 return y;
}
void display()
{
 int i;
 if(front==-1)
 printf("\nQUEUE IS EMPTY\n");
 else
 {
 printf("\nDisplaying Queue:\n");
 for(i=front;i<=rear;i++)
 printf("%d\n",queue[i]);
 }
}
int main()
{
 int choice,x,item;
 printf("\nImplementation of queue using array:\n");
 printf("\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
 do
 {
 printf("\nEnter your choice: ");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1:
printf("\nEnter the value to be inserted: ");
scanf("%d",&x);
enqueue(x);
break;
 case 2:
item=dequeue();
if(item!=0)
```

```
printf("\nThe deleted element is %d",item);
break;
 case 3:
display();
break;
 case 4:
printf("\nEXIT\n");
break;
 default:
printf("\nInvalid Choice! Enter between (0-4)\n");
 }
 }
 while(choice!=4);
  return 0;
}
```

10. CIRCULAR QUEUE

```
#include<stdio.h>
#define size 5
int cqueue[size],front=-1,rear=-1;
void enqueue(int x)
{
 //check for overflow
 if((front==0 && rear==size-1) || (rear==(front-1)))
 {
 printf("\nQUEUE IS FULL\n");
 }
 //inserting first element
 else if(front==-1)
 {
 front=rear=0;
 cqueue[rear]=x;
 }
 //read reached maximum index but free space available in front
 else if(rear==size-1 && front!=0)
 {
 rear=0;
 cqueue[rear]=x;
 }
 else
 {
 rear++;
 cqueue[rear]=x;
 }
}
int dequeue()
{
 int y;
 if(front==-1)
 {
 printf("\nQUEUE IS EMPTY\n");
 return 0;
```

```
 }
 y=cqueue[front];
 //deleting the only element present in queue
 if(front==rear)
 {
 front=-1;
 rear=-1;
 }
 //deleting element at last position in the queue
 else if(front==size-1)
 front=0;
 else
 front++;
 return y;
}
void display()
{
 int i;
 if(front==-1)
 {
 printf("\nQUEUE IS EMPTY\n");
 return;
 }
 printf("\nElements in circular queue are:\n");
 if(rear>=front)
 {
 for(i=front;i<=rear;i++)
 printf("%d",cqueue[i]);
 }
 else
 {
 for(i=front;i<size;i++)
 printf("%d",cqueue[i]);
 for(i=0;i<=rear;i++)
 printf("%d",cqueue[i]);
 }
}
```

```
int main()
{
 int choice,x,item;
 printf("\nImplementation of circular queue using array:\n");
 printf("\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit");
 do
 {
 printf("\nEnter the choice: ");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1:
printf("\nEnter the value to be inserted: ");
scanf("%d",&x);
enqueue(x);
break;
 case 2:
item=dequeue();
if(item!=0)
printf("\nThe deleted element is %d",item);
break;
 case 3:
display();
break;
 case 4:
printf("\nEXIT\n");
break;
 default:
printf("\nInvalid Choice! Enter between (0-4)");
 }
 }
 while(choice!=4);
 return 0;
}
```

11. SLL

```
#include<stdio.h>
#include<stdlib.h>
struct sll
{
 int data;
 struct sll *next;
};
typedef struct sll node;
node *head=NULL,*new,*temp,*prev;
int key;
void create()
{
 if(head==NULL)
 {
 new=(node*)malloc(sizeof(node));
 printf("\nEnter the data: ");
 scanf("%d",&new->data);
 new->next=NULL;
 head=new;
 }
 else
 {
 printf("\nLinked list already exists");
 }
}
```

```
void insert_beg()
{
 new=(node*)malloc(sizeof(node));
 printf("\nEnter the data: ");
 scanf("%d",&new->data);
 new->next=head;
 head=new;
}
void insert_end()
{
 new=(node*)malloc(sizeof(node));
 printf("\nEnter the data: ");
 scanf("%d",&new->data);
 new->next=NULL;
 temp=head;
 while(temp->next!=NULL)
 temp=temp->next;
 temp->next=new;
}
void insert_middle()
{
 printf("\nEnter the key: ");
 scanf("%d",&key);
 new=(node*)malloc(sizeof(node));
 printf("\nEnter the data: ");
 scanf("%d",&new->data);
```

```
 temp=head;
 while(temp->data!=key)
 {
 temp=temp->next;
 }
 new->next=temp->next;
 temp->next=new;
}
void delete_beg()
{
 temp=head;
 head=head->next;
 free(temp);
}
void delete_end()
{
 temp=head;
 while(temp->next!=NULL)
 {
 prev=temp;
 temp=temp->next;
 }
 prev->next=NULL;
 free(temp);
}
void delete_middle()
```

```
{
 temp=head;
 printf("\nEnter the key: ");
 scanf("%d",&key);
 while(temp->data!=key)
 {
 prev=temp;
 temp=temp->next;
 }
 prev->next=temp->next;
 free(temp);
}
void display()
{
 temp=head;
 while(temp!=NULL)
 {
 printf("%d ",temp->data);
 temp=temp->next;
 }
}
int main()
{
 int choice;
 do
 {
```

```
 printf("\n\n1.Create\n2.Insert in beginning\n3.Insert in end");
 printf("\n4.Insert in middle\n5.Delete in beginning\n6.Delete in end");
 printf("\n7.Delete in middle\n8.Display\n9.Exit");
 printf("\nEnter your choice: ");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1:
create();
break;
 case 2:
insert_beg();
break;
 case 3:
insert_end();
break;
 case 4:
insert_middle();
break;
 case 5:
delete_beg();
break;
 case 6:
delete_end();
break;
 case 7:
```

```
delete_middle();
break;
 case 8:
display();
break;
 case 9:
printf("\nEXIT");
break;
 default:
printf("\nEnter a valid choice (1-8)");
 }
 }
 while(choice!=9);
 return 0;
}
```

12. ADDITION OF 2 POLY

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct sll
{
 int c;
 int e;
 struct sll *next;
};
typedef struct sll p;
p *poly1=NULL, *poly2=NULL, *poly3=NULL, *temp;
void push(p *num)
{
 int i=1,term=0;
 while(i==1)
 {
 printf("ENTER THE COEFFICIENT OF TERM %d:->\n",term+1);
 scanf("%d",&num->c);
 printf("ENTER THE EXPONENT OF THE TERM %d:->\n",term+1);
 scanf("%d",&num->e);
 num->next=(p*)malloc(sizeof(p));
 num=num->next;
 num->next=NULL;
 printf("DO YOU WANT TO CONTINE ADDING TERMS :('1=YES' ,
'2=NO')\n");
 scanf("%d",&i);
 term++;
 }
}
void display()
{
 p *num=poly3;
 temp=num;
 while(temp->next!=NULL)
 {
```

```
 if(temp->next->next!=NULL)
 {
 if(temp->next->c<0)
 printf("%dX^%d",temp->c,temp->e);
 else
 printf("%dX^%d+",temp->c,temp->e);
 }
 else
 printf("%dX^%d",temp->c,temp->e);
 temp=temp->next;
 }
}
void addPolynomial(p *p1,p *p2,p *p3)
{
 while(p1->next && p2->next)
 {
 if(p1->e > p2->e)
 {
 p3->e=p1->e;
 p3->c=p1->c;
 p1=p1->next;
 }
 else if(p1->e < p2->e)
 {
 p3->e=p2->e;
 p3->c=p2->c;
 p2=p2->next;
 }
 else
 {
 p3->e=p1->e;
 p3->c=p1->c+p2->c;
 p1=p1->next;
 p2=p2->next;
 }
 p3->next=(p *)malloc(sizeof(p));
 p3=p3->next;
```

```
 p3->next=NULL;
 }
 while(p1->next || p2->next)
 {
 if(p1->next)
 {
 p3->e=p1->e;
 p3->c=p1->c;
 p1=p1->next;
 }
 if(p2->next)
 {
 p3->e=p2->e;
 p3->c=p2->c;
 p2=p2->next;
 }
 p3->next=(p *)malloc(sizeof(p));
 p3=p3->next;
 p3->next=NULL;
 }
}
void main()
{
 poly1=(p *)malloc(sizeof(p));
 poly2=(p *)malloc(sizeof(p));
 poly3=(p *)malloc(sizeof(p));
 printf("ENTER THE 1st POLYNOMIAL :->\n");
 push(poly1);
 printf("ENTER THE 2nd POLYNOMIAL :->\n");
 push(poly2);
 addPolynomial(poly1,poly2,poly3);
 printf("RESULT IS:->\n");
 display();
}
```

13. DLL

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 //Defining the 3 parts of doubly linked list NODE
int data;
struct node *next;
struct node *prev;
};
struct node *head=0, *new1, *temp, *tail;
int count=0;
int display()
{
temp = head;
while (temp != 0)
{
printf("%d ", temp->data);
temp = temp->next;
count++;
}
printf("\n");
printf("The number of nodes that have been created are: %d", count);
count=0;
return 0;
}
void create_Dll()
//Creating the list
{
new1 = (struct node *)malloc(sizeof(struct node));
 //Allocating storage size
printf("enter the data");
scanf("%d", &new1->data);
new1->prev = 0;
new1->next = 0;
head = tail = new1;
```

```
display();
}
void insert_beg()
{
new1 = (struct node *)malloc(sizeof(struct node));
printf("enter the data");
scanf("%d", &new1->data);
new1->prev = 0;
new1->next = 0;
head->prev = new1;
new1->next = head;
head = new1;
display();
}
void insert_end()
{
new1 = (struct node *)malloc(sizeof(struct node));
printf("enter the data");
scanf("%d", &new1->data);
new1->prev = 0;
new1->next = 0;
tail->next = new1;
new1->prev = tail;
tail = new1;
display();
}
void insert_at_pos()
{
int pos, i = 1;
printf("Enter the pos: ");
scanf("%d", &pos);
//if (pos > count)
//{
//printf("Invalid position");
//}
 if (pos == 1)
{
```

```
insert_beg();
}
else
{
temp = head;
new1 = (struct node *)malloc(sizeof(struct node));
printf("enter the data");
scanf("%d", &new1->data);
while (i < pos - 1)
// one less than the position we need to insert at
{
temp = temp->next;
i++;
}
new1->prev = temp;
new1->next = temp->next;
temp->next = new1;
new1->next->prev = new1;
//storing 3rd element
}
display();
}
void insert_after_pos()
{
int pos, i = 1;
printf("Enter the pos: ");
scanf("%d", &pos);
//if (pos > count)
//{
//printf("Invalid position");
//}
//
//{
temp = head;
new1 = (struct node *)malloc(sizeof(struct node));
printf("enter the data");
scanf("%d", &new1->data);
```

```
while (i < pos)
// on the position so we can insert after the pos
{
temp = temp->next;
i++;
}
new1->prev = temp;
new1->next = temp->next;
temp->next = new1;
new1->next->prev = new1;
//storing 3rd element
//}
display();
}
void delete_beg()
{
if (head == 0)
{
printf("Dll is empty");
}
else
{
temp = head;
head = head->next;
head->prev = 0;
free(temp);
}
display();
}
void delete_end()
{
if (tail == 0)
{
printf("Dll is empty");
}
else
{
```

```
temp = tail;
tail = tail->prev;
tail->next = 0;
free(temp);
}
display();
}
void delete_mid()
{
temp = head;
int pos, i = 1;
printf("enter the pos: ");
scanf("%d", &pos);
while (i < pos)
{
temp = temp->next;
i++;
}
temp->prev->next = temp->next;
temp->next->prev = temp->prev;
free(temp);
display();
return ;
}
void reverse()
{
struct node *current, *nextnode;
current = 0;
nextnode = 0;
current = head;
while (current != 0)
{
nextnode = current->next;
current->next = current->prev;
current->prev = nextnode;
current = nextnode;
}
```

```
temp = head;
head = tail;
tail = temp;
display();
}
int main()
{
int choice;
do
{
printf( "\n1. Create\n2. Insert at beginning\n3. Insert at end\n4. Insert at
position\n5.Insert after position");
printf("\n6. Delete at beginning\n7. Delete at end\n8. Delete at
middle\n9. Display");
printf("\n10.reverse the Doubly linked list\n11.EXIT");
printf("\nEnter the choice: ");
scanf("%d", &choice);
switch (choice)
{
case 1:
create_Dll();
break;
case 2:
insert_beg();
break;
case 3:
insert_end();
break;
case 4:
insert_at_pos();
break;
case 5:
insert_after_pos();
break;
case 6:
delete_beg();
break;
```

```
case 7:
delete_end();
break;
case 8:
delete_mid();
break;
case 9:
display();
break;
case 10:
reverse();
break;
case 11:
printf("\nEXIT");
break;
default:
printf("\nEnter a valid Choice(1-11)");
}
} while (choice != 11);
}
```

14. TRANSPOSE OF A TREE

```
#include<stdio.h>
#include<stdlib.h>
struct BinaryTree
{
 struct BinaryTree *left;
 int data;
 struct BinaryTree *right;
};
typedef struct BinaryTree node;
node *root=NULL, *new;
//create tree
node* create(node* root,int item)
{
 if(root==NULL)
 {
 new=(node *)malloc(sizeof(node));
 new->data=item;
 new->left=new->right=NULL;
 root=new;
 }
 else if(item<root->data)
 {
 root->left=create(root->left,item);
 }
 else if(item>root->data)
 {
 root->right=create(root->right,item);
 }
 return root;
}
//Inorder
void inorder(node *root)
{
 if(root)
 {
```

```
 inorder(root->left);
 printf("%d\t",root->data);
 inorder(root->right);
 }
}
//Preorder
void preorder(node *root)
{
 if(root==NULL)
 return;
 printf("%d\t",root->data);
 preorder(root->left);
 preorder(root->right);
}
//Postorder
void postorder(node *root)
{
 if(root==NULL)
 return;
 postorder(root->left);
 postorder(root->right);
 printf("%d\t",root->data);
}
int main()
{
 int choice,item;
 do
 {
 printf("\n1.Create\n2.Inorder Traversal\n3.Preorder
Traversal\n4.Postorder Traversal\n5.EXIT");
 printf("\nEnter your choice: ");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1: printf("Enter the element to be inserted: ");
 scanf("%d",&item);
 root=create(root,item);
```

```
 break;
 case 2: if(root==NULL)
 printf("\nTree is empty.\n");
 else
 {
 printf("\nInorder traversal");
 inorder(root);
 }
 break;
 case 3: if(root==NULL)
 printf("\nTree is empty.\n");
 else
 {
 printf("Preorder Traversal\n");
 preorder(root);
 }
 break;
 case 4: if(root==NULL)
 printf("\nTree is empty.\n");
 else
 {
 printf("Postorder Traversal\n");
 postorder(root);
 }
 break;
 case 5: printf("\nEXIT.\n");
 break;
 default: printf("\nEnter a valid choice(1-5)\n");
 }
 }
 while(choice!=5);
 return 0;
}
```

15. BST

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

struct node
{
   int key;
   struct node *left, *right;
};

struct node *newNode(int item)
{
   struct node *temp = (struct node *)malloc(sizeof(struct node));
   temp->key = item;
   temp->left = temp->right = NULL;
   return temp;
}

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

struct node *insert(struct node *node, int key)
{

   if (node == NULL)
      return newNode(key);

   if (key < node->key)
      node->left = insert(node->left, key);
```

```
    else if (key > node->key)
        node->right = insert(node->right, key);

    return node;
}
struct node *minValueNode(struct node *node)
{
    struct node *current = node;

    while (current && current->left != NULL)
        current = current->left;

    return current;
}

struct node *deleteNode(struct node *root, int key)
{
    if (root == NULL)
        return root;

    if (key < root->key)
        root->left = deleteNode(root->left, key);

    else if (key > root->key)
        root->right = deleteNode(root->right, key);

    else
    {

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

```
            struct node *temp = root->left;
            free(root);
            return temp;
        }

        struct node *temp = minValueNode(root->right);

        root->key = temp->key;

        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}

int main()
{
    struct node *root = NULL;
    int op;
    int n;
    int k;
        printf("ENTER THE NO OF ELEMENTS IN THE TREE:");
    scanf("%d", &n);
    while (n--)
    {
        printf("ENTER THE KEY:");
        scanf("%d", &k);
        root = insert(root, k);
    }

    while (op != 4)
    {
        printf("1.insert\n2.delete\n3.INORDER TRAVERSAL.\nENTER
OPTION:");
        scanf("%d", &op);
        switch (op)
        {
        case 1:
```

```
            printf("ENTER THE KEY:");
            scanf("%d", &k);
            insert(root, k);
            break;
        case 2:
            printf("ENTER THE KEY TO BE DELETED:");
            scanf("%d", &k);
            root = deleteNode(root, k);
            inorder(root);
            break;
        case 3:
            inorder(root);
            break;
        }
    }
    return 0;

    return 0;
}
```

16. BFS N DFS

B F S

```
#include<stdio.h>
int A[10][10],visited[10],queue[10],f=-1,r=-1,n;
void BFS(int v)
{
 int u;
 f=0;
 queue[++r]=v;
 visited[v]=1;
 printf("\n%d",v);
 while(f<=r)
 {
 v=queue[f++];
 for(u=0;u<n;u++)
 {
 if((visited[u]==0) && (A[v][u]==1))
 {
 printf("\n%d",u);
 queue[++r]=u;
 visited[u]=1;
 }
 }
 }
}
int main( )
{
 int i,j,source;
 printf("\nEnter the number of vertices: ");
 scanf("%d",&n);
 printf("\nEnter adjecency matrixof the graph:\n");
 for(i=0;i<n;i++)
 {
 for(j=0;j<n;j++)
 {
```

```
 scanf("%d",&A[i][j]);
 }
 }
 //visited is initialised to zero
 for(i=0;i<n;i++)
 {
 visited[i]=0;
 }
 printf("\nEnter the source vertex: ");
 scanf("%d",&source);
 BFS(source);
}
```

D F S

```
#include<stdio.h>
#include<conio.h>
int A[10][10],visited[10],n;
void DFS(int v)
{
 int u;
 printf("\n%d",v);
 visited[v]=1;
 for(u=0;u<n;u++)
 {
 if((visited[u]==0) && (A[v][u]==1))
 DFS(u);
 }
}
int main()
{
 int i,j,source;
 printf("\nEnter the number of vertices: ");
 scanf("%d",&n);
 printf("\nEnter the adjecency matrix of the graph:\n");
 for(i=0;i<n;i++)
 {
```

```
 for(j=0;j<n;j++)
 {
 scanf("%d",&A[i][j]);
 }
 }
 //visited is initialised to zero
 for(i=0;i<n;i++)
 {
 visited[i]=0;
 }
 printf("\nEnter the source vertix: ");
 scanf("%d",&source);
 DFS(source);
}
```