1.Write a program to read two polynomials and store them in an array. Calculate the sum of the two polynomials and display the first polynomial, second polynomial and the resultant polynomial.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct poly
{
 int coef;
 int exp;
};
struct poly a[10],b[10],c[10];
int d1,d2,d3;
int polyAdd(struct poly p1[],struct poly p2[],struct poly p3[],int d1,int d2)
{
 int i,j,m=0;
 if(d1==d2)
 {
  for(i=0;i<=d1;++i)
  {
   p3[m].coef=p1[i].coef+p2[i].coef;
   p3[m].exp=p1[i].exp;
   ++m;
  }
 }
 else if(d1>d2)
 {
  for(i=0;i<=d2;i++)
 {
  p3[m].coef=p1[i].coef+p2[i].coef;
  p3[m].exp=p2[i].exp;
  ++m;
 }
 for(i=d2+1;i<=d1;++i)
 {
  p3[m].coef=p1[i].coef;
  p3[m].exp=p1[i].exp;
  ++m;
 }
}
else
{
 for(i=0;i<=d1;i++)
 {
  p3[m].coef=p1[i].coef+p2[i].coef;
  p3[m].exp=p1[i].exp;
  ++m;
 }
 for(i=d1+1;i<=d2;++i)
 {
  p3[m].coef=p2[i].coef;
  p3[m].exp=p2[i].exp;
  ++m;
 }
 return m;
}
```

```
}
void display(struct poly p[],int n)
{
 int i;
 for(i=0;i<n;i++)
 {
   if(p[i].coef!=0)
     printf("+ %dx^%d ",p[i].coef,p[i].exp);
 }
}
void main()
{
   int ch,i;
   printf("\n\t\t POLYNOMIAL ADDITION");
   printf("\n\nEnter the degrees of the two polynomials : ");
   scanf("%d%d",&d1,&d2);
   printf("\nEnter the polynomial 1(Coefficient of x^0 , x^1...) : ");
   for(i=0;i<=d1;++i)
   {
     scanf("%d",&a[i].coef);
     a[i].exp=i;
   }\
   printf("\nEnter the polynomial 2(Coefficient of x^0 , x^1.....) : ");
   for(i=0;i<=d2;++i)
   {
     scanf("%d",&b[i].coef);
     b[i].exp=i;
   }
```

```
    d3=polyAdd(a,b,c,d1,d2);
    printf("\n First Polynomial is : ");
    for(i=0;i<=d1;i++)
    {
     if(a[i].coef!=0)
     printf(" +%dx^%d ",a[i].coef,a[i].exp);
    }
    printf("\n Second Polynomial is : ");
    for(i=0;i<=d1;i++)
    {
     if(b[i].coef!=0)
     printf(" +%dx^%d ",b[i].coef,b[i].exp);
    }
    printf("\n Resultant Polynomial is  : ");
    display(c,d3);
}
```

2. C Write a program to enter two matrices in normal form. Write a function to convert two matrices to sparse matrix and display it. Also find the transpose of the two sparse matrices display it. Find the sum of the two sparse matrices display the sum.

```
//Sparse Matrix Addition
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int
a[10][10],b[10][10],sa[10][10],sb[10][10],sc[10][10],trans1[10][10],trans2[10][10]
,m,n,p,q;
```

```
int max1=0,max2=0,max3;
int readM(int mat[10][10],int r,int c) // Function to read matrix
{
 int i,j;
 for(i=0;i<r;++i)
 {
  for(j=0;j<c;++j)
  {
   scanf("%d",&mat[i][j]);
  }
 }
}
int toSparse(int mat[10][10],int sparse[10][10],int max,int r,int c)   // Function
{                                                              to convert to sparse matrix
 int i,j,k;                              //   Sparse Matrix is represented as :-
 sparse[0][0]=r;                         //    ROW    COL    VALUE
 sparse[0][1]=c;                         //     -      -       -
 k=1;                                    //     -      -       -
 for(i=0;i<r;++i)
 {
  for(j=0;j<c;++j)
  {
   if(mat[i][j]!=0)
   {
    sparse[k][0]=i;
    sparse[k][1]=j;
    sparse[k][2]=mat[i][j];
    ++k;++max;
   }
  }
 }
 sparse[0][2]=k-1;
 return max;
}
void display(int mat[10][10],int max,int c)  // Function to display sparse matrix
{
 int i,j;
 for(i=0;i<=max;++i)
 {
  for(j=0;j<c;++j)
  {
   printf("%d ",mat[i][j]);
  }
  printf("\n");
 }
}
int add(int sc[10][10],int sa[10][10],int sb[10][10])  // Function to add Sparse
 {                                                               matrix
    int t1,t2,i,j,k;
    i=j=k=0;
   if((sa[0][0]!=sb[0][0])||(sa[0][1]!=sb[0][1]))
    {
      printf("\n Sparse matrix cannot be added");


    }
    else
```

```
{
  t1=sa[0][2];
  t2=sb[0][2];
  sc[0][0]=sa[0][0];
  sc[0][1]=sa[0][1];
  while(i<=t1&&j<=t2)
  {
        if(sa[i][0]<sb[j][0])
        {
              sc[k][0]=sa[i][0];
              sc[k][1]=sa[i][1];
              sc[k][2]=sa[i][2];
              k++;i++;
        }
        else if(sa[i][0]>sb[j][0])
        {
              sc[k][0]=sb[j][0];
              sc[k][1]=sb[j][1];
              sc[k][2]=sb[j][2];
              k++;j++;
        }
        else if(sa[i][1]<sb[j][1])
        {
          sc[k][0]=sa[i][0];
          sc[k][1]=sa[i][1];
          sc[k][2]=sa[i][2];
          i++;k++;
        }
        else if(sa[i][1]>sb[j][1])
        {
          sc[k][0]=sb[j][0];
          sc[k][1]=sb[j][1];
          sc[k][2]=sb[j][2];
          k++;j++;
        }
        else
        {
                sc[k][0]=sa[i][0];
                sc[k][1]=sa[i][1];
                sc[k][2]=sa[i][2]+sb[j][2];
                k++;i++;j++;
        }
       }
       while(i<=t1)
       {
                sc[k][0]=sa[i][0];
                sc[k][1]=sa[i][1];
                sc[k][2]=sa[i][2];
                k++;i++;
   }
       while(j<=t2)
       {
           sc[k][0]=sb[j][0];
                sc[k][1]=sb[j][1];
                sc[k][2]=sb[j][2];
                k++;j++;
```

```
            }
            sc[0][2]=k-1;
        }
        return (k-1);
    }
    void transpose(int mat[10][10],int trans[10][10])   // Function to find the
                                                         transpose of sparse matrix
    {
        int i,j,k,n;
        trans[0][0]=mat[0][1];
        trans[0][1]=mat[0][0];
        trans[0][2]=mat[0][2];
        k=1;
        n=mat[0][2];
        for(i=0;i<mat[0][1];i++)
        {
          for(j=1;j<=n;j++)
          {
               if(i==mat[j][1])
               {
                     trans[k][0]=i;
                     trans[k][1]=mat[j][0];
                     trans[k][2]=mat[j][2];
                     k++;


               }
            }
        }
```

```
}
void main()
{
  printf("\nEnter the rows and column of matrix 1: ");
  scanf("%d%d",&m,&n);
  printf("\nEnter the matrix 1 : ");
  readM(a,m,n);
  printf("\nEnter the rows and column of matrix 2: ");
  scanf("%d%d",&p,&q);
  printf("\nEnter the matrix 2 : ");
  readM(b,p,q);
  max1=toSparse(a,sa,max1,m,n);
  max2=toSparse(b,sb,max2,p,q);
  printf("\nSparse matrix 1 is :-\n");
  display(sa,max1,n);
  printf("\nSparse matrix 2 is :-\n");
  display(sb,max2,q);
  printf("\n Transpose of sparse matrix 1 :-\n");
  transpose(sa,trans1);
  display(trans1,max1,n);
  printf("\n Transpose of sparse matrix 2 :-\n");
  transpose(sb,trans2);
  display(trans2,max2,q);
  printf("\n Sum of sparse matrix1 and matrix2  : -\n");
  max3=add(sc,sb,sa);
  display(sc,max3,n);
 }
```

3. Implement a circular queue using arrays with the operations:

1. **Insert an element to the queue.**
2. **Delete a elements from the queue.**
3. **Display the contents of the queue after each operation**

```
//Circular Queue..
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 5
int cq[size];
int front=0,rear=0;
void display()
{
 int i;
 printf("\n The circular queue is : ");
 if(front<=rear)
 {
  for(i=front;i<=rear;i++)
   printf(" %d ",cq[i]);
 }
 else
 {
  for(i=front;i<=size;i++)
   printf(" %d ",cq[i]);
  for(i=1;i<=rear;i++)
   printf(" %d ",cq[i]);
 }
}
void enq(int item)
```

```
{
 if(front==((rear%size)+1))
  printf("\n Queue is full");
 else
 {
  if(front==0)
  {
   front=rear=1;
   cq[rear]=item;
  }
  else
  {
   int next=(rear%size)+1;
   if(next!=front)
   {
    rear=next;
    cq[rear]=item;
   }
  }
 }
 display();
}
void deq()
{
 if(front==0)
  printf("\n Queue is empty");
 else
 {
   int item=cq[front];
   if(front==rear)
    front=rear=0;
   else
    front=(front%size)+1;
   printf("\n Deleted %d",item);
  }
  display();
 }
 void main()
 {
  int item,m;
  printf("\n\t\tCIRCULAR QUEUE");
  while(1)
  {
   printf("\n\n 1.Enqueue\n 2.Dequeue\n 3.Display\n 4.Exit");
   printf("\n Enter your choice : ");
   scanf("%d",&m);
   switch(m)
   {
    case 1 :if(front==((rear%size)+1))
              printf("\n Queue if full");
              else
             {
              printf("\n Enter the item to be enqueued : ");
              scanf("%d",&item);
               enq(item);
             }
```

```
            break;
    case 2 :deq();break;
    case 3 :display();break;
    case 4 :exit(1);
    default:printf("\n Invalid entry");
  }
 }
}
```

4. Implement a Double-Ended Queue (DEQUEUE) with the operations:

1. **Insert elements to the Front of the queue.**
2. **Insert elements to the Rear of the queue**
3. **Delete elements from the Front of the queue.**
4. **Delete elements from the Rear of the queue.**
5. **Display the queue after each operation.**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 5
int dq[size];
int front=0,rear=0;
void display()
{
 if(front==0)
  printf("\n queue is empty");
 else
 {
  int i;
  printf("\n The double ended queue is : ");
  if(front<=rear)
  {
   for(i=front;i<=rear;i++)
    printf(" %d ",dq[i]);
  }
  else
  {
   for(i=front;i<=size;i++)
    printf(" %d ",dq[i]);
   for(i=1;i<=rear;i++)
    printf(" %d ",dq[i]);
  }
 }
}
void pushdq(int item)
{
 if(((front==1)&&(rear==size))||(front==rear+1))
  printf("\n Overflow");
 else
 {
  if(front==0)
   front=rear=1;
  else if(front==1)
   front=size;
  else
  {
    --front;
   }
    dq[front]=item;
  }
   display();
 }
 void popdq()
 {
  if(front==0)
   printf("\n Cannot delete value at front end");
  else
  {
   int item=dq[front];
   if(front==rear)
    front=rear=0;
   else if(front==size)
    front=1;
   else
    front=front+1;
   printf("\n Deleted %d ",item);
  }
   display();
 }
 void inject(item)
 {
  if(((front==1)&&(rear==size))||(front==rear+1))
   printf("\n Cannot insert value at rear end");
  else
  {
   if(front==0)
    front=rear=1;
   else if(rear==size)
    rear=1;
   else
   {
    ++rear;
   }
    dq[rear]=item;
  }
  display();
 }
 void eject()
 {
  if(front==0)
   printf("\n Cannot delete value at rear end");
  else
  {
   int item=dq[rear];
   if(front==rear)
    front=rear=0;
   else if(rear==1)
    rear=size;
   else
    --rear;
   printf("\n Deleted %d ",item);
  }
  display();
 }

 void main()
 {
  int item,m;
  printf("\n\t\tDOUBLE ENDED QUEUE");
  while(1)
  {
   printf("\n\n 1.Push DQ\n 2.Pop DQ\n 3.Inject\n 4.Eject\n 5.Display\n
   6.Exit");
   printf("\n Enter your choice : ");
   scanf("%d",&m);
   switch(m)
   {
    case 1 :if(((front==1)&&(rear==size))||(front==rear+1))
             printf("\n Cannot insert item at front");
           else
          {
           printf("\n Enter the item to be enqueued : ");
           scanf("%d",&item);
           pushdq(item);
          }
          break;
    case 2 :popdq();break;
    case 3 : if(((front==1)&&(rear==size))||(front==rear+1))
               printf("\n Cannot insert item at rear");
              else
            {
             printf("\n Enter the item to be enqueued : ");
            scanf("%d",&item);
           inject(item);
           }
           break;
      case 4 :eject();
            break;
      case 5 :display();break;
      case 6 :exit(1);
      default:printf("\n Invalid entry");
     }
  }
}
```

5. Write a menu driven program for performing the following operations on a Linked List:

1.Display

2.Insert at Beginning

3.Insert at End

4.Insert at a specified Position

5.Delete from Beginning

6.Delete from End

7.Delete from a specified Position

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

```
struct node
{
 int data;
 struct node *link;
};
struct node *temp,*ptr,*ptr1;
struct node *head;
void addNodeBeg()
{
 int item;
 printf("\n Enter the item you want to insert : ");
 scanf("%d",&item);
 temp=(struct node*)malloc(sizeof(struct node));
 if(temp==NULL)
 {
  printf("\n Cannot create node");
 }
 else
 {
   temp->data=item;
   temp->link=head->link;
   head->link=temp;
 }
 printf("\nNode inserted at beginning.");
}
void addNodeEnd()
{
 int item;
 printf("\n Enter the item you want to insert : ");
 scanf("%d",&item);
 temp=(struct node*)malloc(sizeof(struct node));
 if(temp==NULL)
 {
  printf("\nCannot create new node ");
 }
 else
 {
  if(head->link==NULL)
  {
   temp->data=item;
   temp->link=head->link;
   head->link=temp;
  }
  else
  {
   ptr=head->link;
   while(ptr->link!=NULL)
     ptr=ptr->link;
   temp->link=ptr->link;
    temp->data=item;
  ptr->link=temp;
  }
}
 printf("\nNode inserted at end");
}
void addNodeAny()
```

```
{
 int item,key;
 printf("\n Enter the item you want to insert : ");
 scanf("%d",&item);
 temp=(struct node*)malloc(sizeof(struct node));
 if(head->link==NULL)
 {
        temp->data=item;
        temp->link=head->link;
        head->link=temp;
 }
 else
 {
  printf("\n Enter the data after you want to insert the node : ");
  scanf("%d",&key);
  ptr=head->link;
  while((ptr->data!=key)&&(ptr->link!=NULL))
    ptr=ptr->link;
  temp->link=ptr->link;
  temp->data=item;
  ptr->link=temp;
  printf("\nNode inserted");
 }
}
void delNodeBeg()
{
 if(head->link==NULL)
  printf("\nUnderfow");
 else
 {
  temp=head->link;
  head->link=temp->link;
  free(temp);
  printf("\nNode deleted from begining");
 }
}
void delNodeEnd()
{
 if(head->link==NULL)
   printf("\nUnderfow");
 else
 {
  ptr=head;
  ptr1=head->link;
  while(ptr1->link!=NULL)
  {
   ptr=ptr->link;
   ptr1=ptr1->link;
  }
  ptr->link=NULL;
 }
 free(ptr1);
 printf("\nNode deleted froom the end");
}
void delNodeAny()
{
 int key;
 if(head->link==NULL)
  printf("\nUnderfow");
 else
 {
  printf("\n Enter the data of node you want to delete : ");
  scanf("%d",&key);
  ptr=head;
  ptr1=head->link;
  while(ptr1->data!=key)
  {
   ptr=ptr->link;
   ptr1=ptr1->link;
  }
  ptr->link=ptr1->link;
 }
 free(ptr1);
 printf("\nNode deleted");
}
void display()
{
 if(head->link==NULL)
  printf("\n Underflow...Empty linked list");
 else
 {
  printf("\n The linked list is:- \n\t");
  ptr=head->link;
  while(ptr!=NULL)
  {
   printf(" %d ",ptr->data);
   ptr=ptr->link;
  }
 }
}
void main()
{
 int i;
 head=(struct node*)malloc(sizeof(struct node));
 head->link=NULL;
 printf("\n\t\tSINGLE LINKED LIST");
 while(1)
 {
  printf(" \n\n 1.Insert at beg \n 2.Insert at end \n 3.Insert at anywhere \n
4.Delete at begening \n 5.Delete from end \n 6.Delete from anywhere \n
7.Traverse \n 8.Exit  ");
  printf(" \n  Enter your choice  ; ");
  scanf("%d",&i);
  switch(i)
  {

   case 1 : addNodeBeg();
            break;

   case 2 : addNodeEnd();
            break;
   case 3 : addNodeAny();
            break;

   case 4 : delNodeBeg();
            break;

   case 5 : delNodeEnd();
            break;

   case 6 : delNodeAny();
            break;

   case 7 : display();
            break;
   case 8 : exit(1);
   default: printf("\nInvalid entry");
  }
 }
}
```

6. Implement a stack using linked list with the operations:

   1.Push elements to the queue.

   2.Pop elements from the queue.

   3.Display the queue after each operation.

```
//Stack using linked list
#include<stdio.h>
#include<conio.h>
#include<stdlib.h0>
```

```
struct node
{
 int data;
 struct node *link;
};
struct node *temp;
struct node *top=NULL;
void push(int item)
{
 temp=(struct node*)malloc(sizeof(struct node));
 if(temp==NULL)
  printf("Cannot create node");
 else
 {
  if(top==NULL)
  {
   temp->data=item;
   temp->link=NULL;
   top=temp;
  }
  else
  {
   temp->data=item;
   temp->link=top;
   top=temp;
  }
 }
}
```

```
void pop()
{
 if(top==NULL)
  printf("\n Underflow");
 else
 {
  temp=top;
  printf("Deleted %d",temp->data);
  top=top->link;
  free(temp);
 }
}
void display()
{
 struct node *t;
 if(top==NULL)
  printf(" Underflow");
 else
 {
  t=top;
  printf("\n The Stack is : ");
  while(t!=NULL)
  {
   printf(" %d ",t->data);
   t=t->link;
  }
 }
}
```

```
void main()
{
 int m,i,item;
 printf("\n\t\t STACK USING LINKED LIST");
 while(1)
 {
  printf("\n 1.Push \n 2.Pop\n 3.Traverse\n 4.Exit \n Enter the option : ");
  scanf("%d",&m);
  switch(m)
  {
   case 1 : printf("\n Enter the item to be inserted : ");
            scanf("%d",&item);
            push(item);
            break;
   case 2 : pop();
            break;
   case 3 : display();
            break;
   case 4 : exit(1);
   default: printf("\n Invalid entry");
  }
 }
}
```

7. .Implement a Queue using linked list with the operations:

1.Insert an elements to the queue.

2.Delete an elements from the queue.

3.Display the queue after each operation.

```
//Queue using linked list
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *link;
};
struct node *front=NULL,*rear=NULL,*temp;
void enq(int item)
{
 temp=(struct node*)malloc(sizeof(struct node));
 if(temp==NULL)
  printf("\n Cannot create node");
 else
 {
  if(rear==NULL)
  {
   temp->data=item;
   temp->link=NULL;
   front=rear=temp;
  }
  else
  {
   temp->data=item;
   temp->link=NULL;
   rear->link=temp;
   rear=temp;
  }
 }
}
void deq()
{
 if(front==NULL)
  printf("\n Underflow");
 else
 {
  temp=front;
  printf("\n Deleted %d ",front->data);
  if(front==rear)
   front=rear=NULL;
  else
   front=front->link;
  free(temp);
 }
}
void display()
{
 struct node *ptr;
 if(front==NULL)
  printf("\n Underflow");
 else
 {
  printf("\n The queue is  ; ");
  ptr=front;
  while(ptr!=NULL)
  {
   printf(" %d ",ptr->data);
   ptr=ptr->link;
  }
 }
}
void main()
{
 int m,i,item;
 printf("\n\t\t QUEUE USING LINKED LIST");
 while(1)
 {
  printf("\n 1.Enqueue\n 2.Dequeue\n 3.Display\n 4.Exit\n Enter your choice : ");
  scanf("%d",&m);
  switch(m)
  {
   case 1 : printf("\n Enter the item to be inserted : ");
            scanf("%d",&item);
            enq(item);
            break;
   case 2 : deq();break;
   case 3 : display();break;
   case 4 : exit(1);
   default: printf("\n Invalid entry");
  }
 }
}
```

8. Write a program to reverse the content of queue using stack

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 5
int stack[size],queue[size];
int top=-1,front=-1,rear=-1;
void push(int item)
{
        if(top==(size-1))
         printf("\n Overflow");
        else
    {
        ++top;
                stack[top]=item;
        }
}
int pop()
{
        int item;
        if(top==-1)
        {
         printf("\n Underflow");
         return 0;
    }
        else
        {
          item=stack[top];
         --top;
        return item;
        }
}
void enq(int item)
{
        if(rear==(size-1))
         printf("\n Overflow");
        else
        {
                if(rear==-1)
                 front=rear=0;
                else
                 ++rear;
                queue[rear]=item;
         }
}
int deq()
{
        int item;
        if(front==-1)
        {
          printf("\n Underflow");     return 0;
        }
        else
        {
          item=queue[front];
          if(front==rear)
          front=rear=-1;
         else
          ++front;
         return item;
        }
}
void display(int queue[])
{
        int i;
        if(rear==-1)
         printf("\n Underflow");
        else
        {
          for(i=front;i<=rear;i++)
           printf(" %d ",queue[i]);
         }
}
void main()
{
        int i,x,item;
        printf("\n Enter the elements in queue : ");
        for(i=0;i<size;i++)
        {
                scanf("%d",&item);
                enq(item);
        }
        printf("\n The Queue before reversing its contents : ");
        display(queue);
```

```
	for(i=0;i<size;i++)
	{
	  x=deq();
	  push(x);
	}
	for(i=0;i<size;i++)
	{
		x=pop();
		enq(x);
	}
	printf("\n The Queue after reversing its contents : ");
	display(queue);
}
```

9. Create a Doubly Linked List from a string taking each character from the string. Check if the given string is palindrome in an efficient method.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
struct node
{
	char data;
	struct node *rlink,*llink;
};
struct node *head,*temp,*ptr,*start,*last;
char str[10];
void addNode(char item)
```

```
{
        temp=(struct node*)malloc(sizeof(struct node));
        if(temp==NULL)
         printf("\n Cannot create node");
        else
        {
                temp->data=item;
                if(head->rlink==NULL)
                {
                  head->rlink=temp;
                  temp->llink=head;
                  temp->rlink=NULL;
                }
                else
                {
                        ptr=head->rlink;
                        while(ptr->rlink!=NULL)
                         ptr=ptr->rlink;
                        ptr->rlink=temp;
                        temp->llink=ptr;
                        temp->rlink=NULL;
            }
        }

}
int checkPalindrome()
{
        if(head->rlink==NULL)
        {
          printf("\n Underflow");
          return 0;
    }
        else
        {
                ptr=head->rlink;
                while(ptr->rlink!=NULL)
                 ptr=ptr->rlink;
                start=head->rlink;
                last=ptr;
                while(start!=last)
                {
                        if(start->data!=last->data)
                        {
                                return 0;
                        }
                        else if(last==start->llink)
                         return 1;
                        start=start->rlink;
                        last=last->llink;
                }
          return 1;
        }
}

void main()
{
        head=(struct node*)malloc(sizeof(struct node));
        head->rlink=NULL;
        head->llink=NULL;
        char ch,a;
        int i=0,n,flag;
        printf("\n Enter the string : ");
        gets(str);
        n=strlen(str);
        for(i=0;i<n;i++)
        {
                addNode(str[i]);
        }
        flag=checkPalindrome();
        if(flag==1)
         printf("\n %s is a palindrome",str);
        else
         printf("\n %s not a palindrome",str);
}
```

10. Implementation of searching algorithms – linear search, binary search.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int arr[10],size;
void linear(int arr[],int size,int item)
{
        int i,flag=0,pos;
        for(i=0;i<size;i++)
        {
                if(arr[i]==item)
                {
                        pos=i;
                        flag=1;
                        break;
                }
        }
        if(flag==1)
         printf("\n Item found at positon %d",pos+1);
        else
         printf("\n Item not found");
}
void binary(int arr[],int size,int item)
{
        int low,high,mid,pos,flag=0;
        low=0;
        high=size-1;
        while(low<=high)
        {
                mid=(low+high)/2;
                if(arr[mid]==item)
                {
                        pos=mid;
                        flag=1;
                        break;
                }
                else if(item<arr[mid])
                high=mid-1;
               else
                high=mid+1;
        }
        if(flag==1)
         printf("\n Item found at position %d",pos+1);
        else
         printf("\n Item not found");
}
void main()
{
        int item,i,ch;
        printf("\n Enter the size of the array :");
        scanf("%d",&size);
        printf("\n Enter the elemnts in the array  : ");
        for(i=0;i<size;i++)
         scanf("%d",&arr[i]);
        printf("\n Enter the item to be searched for : ");
        scanf("%d",&item);
        while(1)
        {
        printf("\n Enter 1 for Linear search and 2 for Binary search and 3 to
        Exit: ");
        scanf("%d",&ch);
     switch(ch)
        {
                case 1 : linear(arr,size,item); break;
                case 2 : binary(arr,size,item); break;
                case 3 : exit(1);
                default: printf("\n Invalid entry");
            }
    }
}
```

11. Implementation of sorting algorithms – bubble, insertion, selection, quick, merge sort

```
//Bubblesort
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int arr[10],n;
void display()
{
        int i;
  printf("\n\nSorted array is :-  ");
  for(i=0;i<n;i++)
  {
   printf(" %d",arr[i]);
  }
}
void bubble(int arr[],int n)
{
 int i,j,temp;
 for(i=0;i<(n-1);i++)
 {
  for(j=0;j<(n-i-1);j++)
  {
   if(arr[j]>arr[j+1])
   {
    temp=arr[j];
    arr[j]=arr[j+1];
    arr[j+1]=temp;
   }
  }
 }
}
void insertion(int arr[],int n)
{
  int i,j,temp;
  for(i=1;i<n;i++)
  {
    j=i;
    while(j>0)
    {
    if(arr[j-1]>arr[j])
    {
     temp=arr[j];
     arr[j]=arr[j-i];
     arr[j-i]=temp;
    }
     --j;
    }
  }
}
void selection(int arr[],int n)
```

```
{
  int i,j,min,temp;
  for(i=0;i<n-1;i++)
  {
   min=i;
   for(j=i+1;j<n;j++)
   {
    if(arr[min]>arr[j])
     min=j;
   }
   if(min!=i)
   {
    temp=arr[i];
    arr[i]=arr[min];
    arr[min]=temp;
   }
  }
}
void merge(int arr[],int i1,int j1,int i2,int j2)
{
 int temp[50];
 int i,j,k;
 i=i1;
 j=i2;
 k=0;
 while(i<=j1&&j<=j2)
 {
  if(arr[i]<arr[j])
```

```
   temp[k++]=arr[i++];
  else
   temp[k++]=arr[j++];
 }
 while(i<=j1)
  temp[k++]=arr[i++];
 while(j<=j2)
  temp[k++]=arr[j++];
 for(i=i1,j=0;i<=j2;i++,j++)
  arr[i]=temp[j];
}
void mergesort(int arr[],int i,int j)
{
 int mid;
 if(i<j)
 {
  mid=(i+j)/2;
  mergesort(arr,i,mid);
  mergesort(arr,mid+1,j);
  merge(arr,i,mid,mid+1,j);
 }
}
void quick(int arr[],int low,int high)
{
 int pivot,i,j,temp;
 if(low<high)
 {
  pivot=low;
  i=low;
  j=high;
  while(i<j)
  {
   while((arr[i]<=arr[pivot])&&(i<=high))
    i++;
   while((arr[j]>arr[pivot])&&(j>=low))
    j--;
   if(i<j)
   {
    temp=arr[i];
    arr[i]=arr[j];
    arr[j]=temp;
   }
  }
  temp=arr[j];
  arr[j]=arr[pivot];
  arr[pivot]=temp;
  quick(arr,low,j-1);
  quick(arr,j+1,high);
 }
}
void main()
{
 int i,j,ch;
 printf("\n Enter the size of the array : ");
 scanf("%d",&n);
 printf("\n Enter the elements in array : ");
 for(i=0;i<n;i++)
 {
  scanf("%d",&arr[i]);
 }
 while(1)
 {
   printf("\n\n 1. Bubble sort\n 2. Insertion sort\n 3. Selection sort \n 4. Merge
   sort\n 5. Quick sort\n 6. Exit\n Enter your choice : ");
   scanf("%d",&ch);
   switch(ch)
   {
        case 1 : bubble(arr,n); display(); break;
        case 2 : insertion(arr,n); display(); break;
        case 3 : selection(arr,n); display(); break;
        case 4 : mergesort(arr,0,n-1); display(); break;
        case 5 : quick(arr,0,n-1); display();break;
        case 6 : exit(1);
        default: printf("\n Invalid entry");
   }
 }
}
```

12. Implementation of BFS and DFS for each graph representations.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 20
int q[size],stack[size],a[size][size],vis[size],top=-1,front=-1,rear=-1;
```

```
void enq(int item)
{
        if(rear==(size-1))
         printf("\n Queue is full");
        else
        {
                if(rear==-1)
                 front=rear=0;
                else
                 ++rear;
                q[rear]=item;
         }
}
int deq()
{
        int item;
        if((front==-1)||(front>rear))
          return 0;
        else
        {
          item=q[front];
          ++front;
          return item;
        }
}
void push(int item)
{
        if(top==(size-1))
```

```
          printf("\n Stack is full");
         else
         {
            ++top;
            stack[top]=item;
         }
}
int pop()
{
          int item;
          if(top==-1)
           return 0;
          else
          {
            item=stack[top];
            --top;
            return item;
          }

}
void bfs(int s,int n)
{
         int p,i;
         vis[s]=1;
         enq(s);
         p=deq();
         if(p!=0)
          printf(" %d ",p);
```

```
        while(p!=0)
        {
          for(i=1;i<=n;i++)
          {
                if((a[p][i]==1)&&(vis[i]==0))
                {
                        enq(i);
                        vis[i]=1;
                }
      }
          p=deq();
          if(p!=0)
      printf(" %d ",p);
        }
        for(i=0;i<=n;i++)
        {
                if(vis[i]==0)
                 bfs(i,n);
        }
}
void dfs(int s,int n)
{
        int p,i;
        push(s);
        vis[s]=1;
        p=pop();
        if(p!=0)
         printf(" %d ",p);
        while(p!=0)
        {
          for(i=1;i<=n;i++)
          {
                 if((a[p][i]==1)&&(vis[i]==0))
                 {
                   push(i);
                   vis[i]=1;
                 }
                }
                p=pop();
                if(p!=0)
                 printf(" %d ",p);
        }
        for(i=1;i<=n;i++)
        {
                if(vis[i]==0)
                 dfs(i,n);
        }
}
void main()
{
        int n,i,j,s,ch;
        printf("\n\t\tGRAPH BFS And DFS");
        printf("\n\n Enter the number of vertices : ");
        scanf("%d",&n);
        printf("\n Enter the graph in adjacency matriix form : ");
        for(i=1;i<=n;i++)
```

```
    {
        for(j=1;j<=n;j++)
        {
            printf("\n Enter 1 if %d has an edge with %d  , else 0   :",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    printf("\n\n The Adjacency Matrix is :-\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            printf(" %d ",a[i][j]);
        }
        printf("\n");
    }
    while(1)
    {
        for(i=0;i<=n;i++)
        vis[i]=0;
        printf("\n\n 1. BFS\n 2. DFS\n 3. Exit");
        printf("\n Enter your choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
           case 1 :  printf("\n Enter the source vertex : ");
                     scanf("%d",&s);
                      bfs(s,n); break;
```

```
              case 2 : printf("\n Enter the source vertex : ");
                       scanf("%d",&s);
                       dfs(s,n);
                       break;
              case 3 : exit(1);
              default: printf("\n Invalid entry");
            }
        }
}
```

13. Implementation of binary search trees – creation, insertion, deletion, search

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct bnode
{
        int data;
        struct bnode *lchild,*rchild;
};
struct bnode *root=NULL;
void inorder(struct bnode *root)
{
        if(root==NULL)
          return;
        else
        {
                inorder(root->lchild);
                printf(" %d ",root->data);
                inorder(root->rchild);
         }
}
struct bnode* createnewNode(int item)
{
        struct bnode *temp=(struct bnode*)malloc(sizeof(struct bnode));
        temp->data=item;
        temp->lchild=NULL;
        temp->rchild=NULL;
        return temp;
}
struct bnode* insertNode(struct bnode *root,int item)
{
    if(root==NULL)
     root=createnewNode(item);
    else
        {
                if(item<root->data)
                 root->lchild=insertNode(root->lchild,item);
                else
                 root->rchild=insertNode(root->rchild,item);
         }
        return root;
}
struct bnode* findMin(struct bnode *root)
{
    if(root==NULL)
    return 0;
   else if(root->lchild==NULL)
    return root;
   else
         return findMin(root->lchild);
}
struct bnode *deleteNode(struct bnode *root,int item)
{
        struct bnode *temp;
        if(root==NULL)
         return root;
        else if(item<root->data)
         root->lchild=deleteNode(root->lchild,item);
        else if(item>root->data)
         root->rchild=deleteNode(root->rchild,item);
        else
        {
                if(root->lchild==NULL)
                {
                 temp=root->rchild;
                 free(root);
                 return temp;
                }
                else if(root->rchild==NULL)
                {
                        temp=root->lchild;
                        free(root);
                        return temp;
                }

                temp=findMin(root->rchild);
                root->data=temp->data;
                root->rchild=deleteNode(root->rchild,temp->data);
        }
        return root;
}
int search(struct bnode *root,int item)
{
        if(root==NULL)
           return 0;
        else if(root->data==item)
           return 1;
        else if(item<root->data)
          return (search(root->lchild,item));
        else if(item>root->data)
          return (search(root->rchild,item));
}
void main()
{
        int item,c,a;
        printf("\n\t\t BINARY SEARCH TREE");
        while(1)
        {
                printf("\n\n 1. Create first node \n 2.Insert node \n 3.Delete Node\n
                4.Traversal\n 5.Searching\n 6.Exit");
                printf("\n Enter your choice : ");
        scanf("%d",&c);
        switch(c)
        {
          case 1 : printf("\n Enter the value in the root node : ");
                   scanf("%d",&item);
                   root=createnewNode(item);
                   break;
             case 2 : printf("\n Enter the value of node to be inserted in the
                    tree : ");
                    scanf("%d",&item);
                   root=insertNode(root,item);
                   break;
             case 3 : printf("\n Enter the value to be deleted from the node  :
                     ");
                   scanf("%d",&item);
                        root=deleteNode(root,item);
                        break;
             case 4 : printf("\n The tree in Inorder traversal is  : ");
                      inorder(root);
                        break;
             case 5 : printf("\n Enter the item to be searched for : ");
                   scanf("%d",&item);
                        a=search(root,item);
                        if(a==1)
                         printf("\n Element Found");
                        else
                         printf("\n Element not found");
                        break;
                    case 6 : exit(1);
                    default: printf("\n Invalid entry");


              }
        }
}
```