1. Write a program to implement the QuickSort algorithm

```
#include<stdio.h>
void quicksort(int a[],int low,int high)
{
 int i,j,pivot,temp;
 if(low<high)
 {
  pivot=low,i=1,j=high;
  while(i<j)
  {
   while(a[i]<=a[pivot]&&i<j)
   i++;
   while(a[j]>a[pivot])
   j--;
   if(i<j)
   {
    temp=a[i];
    a[i]=a[j];
    a[j]=temp;
   }
  }
  temp=a[j];
  a[j]=a[pivot];
  a[pivot]=temp;
```

```
    quicksort(a,low,j-1);//left
    quicksort(a,j+1,high);
  }
}
int main()
{
  int i,n;
  printf("enter the size");
  scanf("%d",&n);
  int a[n];
  printf("enter the unsorted elements");
  for(i=0;i<n;i++)
  scanf("%d",&a[i]);
  quicksort(a,0,n-1);//calling the function
  printf("sorted list");
  for(i=0;i<n;i++)
  printf("%d ",a[i]);
}
```

2.Write program to implement Mere Sort algorithm

```
  #include<stdio.h>
void mergepass(int a[],int low,int high)
{

  int mid;
  if(low<high)
```

```
  {
    mid=(low+high)/2;
    mergepass(a,low,mid);
    mergepass(a,mid+1,high);
    mergesort(a,low,mid,high);
    }

}
void mergesort(int a[],int low,int mid,int high)
{
  int i ,j, k,b[100];
  i =low;
  j=mid+1;
  k=low;
  while(i<=mid&&j<=high)
  {
    if(a[i]<=a[j])
    {
      b[k]=a[i];
      i++;
    }
    else
    {
      b[k]=a[j];
      j++;
```

```
    }
    k++;
  }
  if(i<=mid)
  {
    for(i=i;i<=mid;i++)
    {
    b[k]=a[i];
    k++;
    }
    }
  else
  {
  for(j=j;j<=high;j++)
    {
      b[k]=a[j];
      k++;}
  }
  for(k=low;k<=high;k++)
    a[k]=b[k];
}
int main()
{
  int n,i;
  printf("enter size of array");
```

```
 scanf("%d",&n);
 int a[n];
 printf("enetr elements in array");
 for(i=0;i<n;i++)
 scanf("%d",&a[i]);
 mergepass(a,0,n-1);//calling the function
 printf("sorted list");
 for(i=0;i<n;i++)
 printf("%d\n",a[i]);


}
```

3. Write program to implement Shell Sort algorithm

```
#include<stdio.h>
void shellSort(int a[],int n)
{
int i,j,gap,temp;  // 33 31 40 8 12 17 25 42
for(gap=n/2;gap>=1;gap=gap/2) //gap=8/2=4
 {
for(j=gap;j<n;j++) //1. j=4,4<8 , 2. j=5,5<8, 3. j=6,6<8 4. j=7,7<8 5. j=8,8<8(false)
   {
    for(i=j-gap;i>=0;i=i-gap) //j=4 1.i=4-4=0, 2. i=0-4=-4 not> 0 // j=5
        {
       if(a[i]>a[i+gap]) //a[0]>a[0+4] = 33>12 - perform swapping
        {
```

```
            temp=a[i];
            a[i]=a[i+gap];
            a[i+gap]=temp;
           }
         } }
 } }
void main()
{
  int n,i;
  printf("\n Enter n : ");
  scanf("%d",&n);
  int a[n];
  printf("\n Enter n values: "); //33, 31, 40, 8, 12, 17, 25, 42
  for(i=0;i<n;i++)
        scanf("%d",&a[i]);
  shellSort(a,n);
  printf("\n The List after Shell Sorting : ");
  for(i=0;i<n;i++)
        printf("%d  ",a[i]);
}
```

4. Conversion of infix to post fix

```
#include<stdio.h>
#include<string.h>
#define size 20
char stack[size];
```

```
int top=-1;
void push(char x)
{
 top++;
 stack[top]=x;
}

char pop()
{
 if(top==-1)
 return -1;
 else
 return stack[top--];
}

int priority(char x)
{
 if(x=='(' &&x==')')
 return 0;
 if(x=='+' &&x=='-')
 return 1;
 if(x=='*'  &&x=='/')
 return 2;
 if(x=='^')
 return 3;
```

```
}

void main()
{
  char exp[size];
  int i;
  char x;
  printf("\n Enter the expression");
  scanf("%s",exp);
  for(i=0;exp[i]!='\0';i++)
  {
    if(isalnum(exp[i]))
    printf("%c",exp[i]);
    else if(exp[i]=='(')
    push(exp[i]);
    else if(exp[i]==')')
    {
      while((x=pop())!='(')
      printf("%c",x);
    }
    else
    {
      while(priority(stack[top])>=priority(exp[i]))
      printf("%c",pop());
      push(exp[i]);
```

```
   }
  }
  while(top!=-1)
  {
   printf("%c",pop());
  }
}
```

5. Evaluation of post fix expression

```
        #include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define size 20
char stack[size];
int top=-1;
void push(char x)
{
  top++;
  stack[top]=x;
}
char pop()
{
  if(top==-1)
  return -1;
  else
  return stack[top--];
```

```
}
void main()
{
  char exp[size];
  int i,val1,val2;
    char x;
    printf("enter the expression:");
    scanf("%s",exp);
    for (i=0;exp[i]!='\0';i++)
    {
      if(isdigit(exp[i]))
      push(exp[i]-48);
      else
      {
        val1=pop();
        val2=pop();
        switch(exp[i])
        {
          case '+':
            push(val2+val1);
            break;
          case '-':
            push(val2-val1);
            break;
          case '*':
```

```
            push(val2*val1);
            break;
          case '/':
            push(val2/val1);
            break;
      }
    }
  }
  while(top!=-1)
  {
    printf("the postfix expression result is %d ",pop());
  }
}
```

6. balancing brackets

```
#include<stdio.h>
char stack[20];
int top=-1;
void push(char a)
{
stack[++top]=a;
}
char pop()
{
return stack[top--];
}
```

```
int main()
{
char a[20],x;
int i,count=1;
scanf("%s",a);
for(i=0;a[i]!='\0';i++)
{
if(a[i]=='('||a[i]=='{'||a[i]=='[')
push(a[i]);
if(a[i]==')'||a[i]=='}'||a[i]==']')
{
if(top==-1)
count=0;
else
{
x=pop();
if(a[i]==')'&&(x=='['||x=='{'))
count=0;
if(a[i]=='}'&&(x=='('||x=='['))
count=0;
if(a[i]==']'&&(x=='{'||x=='('))
count=0;
}
```

```
}
}
if(top>=0)
count=0;
if(count==0)
printf("Unbalanced\n");
else
printf("Balanced\n");
return 0;
}
```

7.Single Linked List operations create,display, insert at end.insert at middle, insert at begin, delete first, delete last,delete middle, search ,reverse, maximum, mimimum sort

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
   int data;
   struct Node *next;
}*first_node=NULL,*current_node;
int sum_of_elements();
int min_element();
void display();
void insert_at_begin(int n);
```

```
void insert_at_end(int n);
void insert(int n);
void insert_after();
void search();
void delete_at_begin();
void delete_at_end();
void delete_after();
int max_element();
int no_of_occurence(int n);
int length();
void main()
{
int option =1,l;
int choice;
   while (option)
   {
      printf ("----------------------------------------\n");
      printf ("     1   -->   insert_at_begin          \n");
      printf ("     2   -->   insert          \n");
      printf ("     3   -->   insert_after          \n");
      printf ("     4   -->   insert_at_end         \n");
      printf ("     5  -->   delete_at_begin         \n");
      printf ("     6  -->   delete_after         \n");
      printf ("     7  -->   delete_at_end         \n");
      printf ("     8  -->   display         \n");
```

```
 printf ("      9   -->    sum          \n");
 printf ("      10   -->    max_element          \n");
 printf ("      11   -->    min_element          \n");
 printf ("      12   -->    no_of_occurence          \n");
  printf ("      13   -->    length          \n");
printf ("----------------------------------------\n");

printf ("Enter your choice\n");
scanf    ("%d", &choice);
switch (choice)
{
case 1:
printf("enter element");
    scanf("%i",&l);
    insert_at_begin(l);
    display();
    break;
case 2:
    printf("enter element");
    scanf("%i",&l);
    insert(l);
    break;
case 3:
    printf("enter element");
    scanf("%i",&l);
```

```
            insert_after(l);
            break;
        case 4:
           printf("enter element");
            scanf("%i",&l);
            insert_at_end(l);
            break;
        case 5:
        delete_at_begin();
        break;
        case 6:
        delete_after();
        break;
        case 7:
         delete_at_end();
         break;
         case 8:
         display();
         break;
         case 9:
         printf("\n sum of elements in linked list is
%d",sum_of_elements());
         break;
         case 10:
         printf("\n max element in linked list is %d",max_element());
```

```
        break;
        case 11:
         printf("\n min element in linked list is %d",min_element());
        break;
        case 12:
        printf("enter element to check occurence");
        scanf("%d",&l);
        printf("no of occurence of %d in linked list is
%d",l,no_of_occurence(l));
        break;
        case 13:
        printf("the length of linked list is %d",length());
        break;
        default:
        printf("\nwrong choice");



       }


       printf ("\nDo you want to continue(Type 0 or 1)?\n");
       scanf    ("%d", &option);
   }
}
```

```
void insert(int n){
  struct Node *new_node;
  new_node=(struct Node*)malloc(sizeof(struct Node));
  new_node->data=n;
  new_node->next=NULL;
  if(first_node==NULL){
     first_node=new_node;
     current_node=new_node;
  }
  else{
     current_node->next=new_node;
     current_node=new_node;
     }

}
void display(){
  struct Node *temp=first_node;
  while(temp!=NULL){
    printf("%i ",temp->data);
    temp=temp->next;
  }
}
void insert_at_begin(int n){
  struct Node *new_node;
  new_node=(struct Node*)malloc(sizeof(struct Node));
```

```
  new_node->data=n;
  new_node->next=first_node;
  first_node=new_node;
}
void insert_at_end(int n){
   struct Node *new_node;
  new_node=(struct Node*)malloc(sizeof(struct Node));
  new_node->data=n;
  new_node->next=NULL;
  current_node->next=new_node;
  current_node=new_node;
}

void insert_after(){
   int key,elem,find=0;
   printf("\nEnter the key");
   scanf("%d",&key);
   struct Node *temp;
   temp=first_node;
   while(temp!=NULL)
   {
      if(temp->data == key)
      {
         find =1;
         break;
```

```
        }
        temp = temp->next;
        }
        if(find==1){
    struct Node *new_node;
   new_node=(struct Node*)malloc(sizeof(struct Node));
printf("\nEnter the element");
    scanf("%d",&elem);
    new_node->data=elem;
    new_node->next = temp->next;
    temp->next= new_node;
        }
        else
        {printf("Key not found");
}
}

void search(){
    int key,find=0,c=0;
    printf("\nEnter the key");
    scanf("%d",&key);
    struct Node *temp;
    temp=first_node;
    while(temp!=NULL)
    {
```

```
        if(temp->data == key)
        {
            find =1;
            break;
        }
        temp = temp->next;
        c++;
        }

        if(find==1){
            printf("\n%d element found at position %d",temp->data,c+1);
        }
        else{
            printf("element not found");
        }
}


void delete_at_begin()
{
    struct Node * temp;
    temp = first_node;
    first_node = first_node->next;
    free(temp);
```

```
}
void delete_at_end(){
    struct Node *temp,*temp1;
    temp=first_node;
    while(temp->next!=NULL){
        temp1=temp;
        temp=temp->next;
    }
    current_node=temp1;
    current_node->next=NULL;
    free(temp);

}

void delete_after(){
    int key,find=0;
    printf("\nEnter the key");
    scanf("%d",&key);
    struct Node *temp,*temp1;
    temp=first_node;
    while(temp!=NULL)
    {
        if(temp->data == key)
        {
            find =1;
```

```
            break;
        }
        temp1=temp;
        temp = temp->next;
        }
    if(find==1){
        temp1->next=temp->next;
        free(temp);
    }
    else{
        printf("\n element not found");
    }

}
int sum_of_elements(){
   struct Node *temp;
   temp=first_node;
   int sum=0;
   while(temp!=NULL){
       sum=sum+temp->data;
       temp=temp->next;
   }
   return sum;
}
int max_element(){
```

```
    struct Node *temp;
    temp=first_node;
    int max=0;
    while(temp!=NULL){
        if(temp->data>max){
            max=temp->data;

        }
        temp=temp->next;
    }
    return max;
}
int min_element(){
    struct Node *temp;
    temp=first_node;
    int min=first_node->data;
    while(temp!=NULL){
        if(temp->data<min){
            min=temp->data;
        }
        temp=temp->next;
    }
    return min;
}
int no_of_occurence(int n){
```

```
    int count=0;
    struct Node *temp;
    temp=first_node;
    while(temp!=NULL){
        if(temp->data==n){
            ++count;
        }
        temp=temp->next;
    }
    return count;
}
int length(){
    int len=0;
    struct Node *temp;
    temp=first_node;
    while(temp!=NULL){
        ++len;
        temp=temp->next;
    }
    return len;
}
/*int insert_middleusingkey()
{
  int key,n,status=0;
  printf("\nEnter the key");
```

```
 scanf("%d",&key);
 struct Node*temp;
 temp=first_node;
 while(temp!=NULL)
 {
  if(temp->data==key)
  {
   status=1;
   break;
  }
  temp=temp->Next;
 }
 if(status==1)
 {
  struct Node*newnode;
  newnode=(struct Node*)malloc(sizeof(struct Node));
  printf("\nEnter the element to insert at middle");
  scanf("%d",&n);
  newnode->data=n;
  newnode->Next=temp->Next;
  temp->Next=newnode;
 }
 else
 printf("Key is not found");
}
```

8. Double Linked List operations create,display, insert at end.insert at middle, insert at begin, delete first, delete last,delete middle, search ,reverse

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
	struct Node*prev;
	int data;
	struct Node*next;
}*firstnode,*currentnode;
void insert_create(int x)
{
	struct Node*newnode;
	newnode=(struct Node*)malloc(sizeof(struct Node));
	newnode->data=x;
	newnode->prev=NULL;
	newnode->next=NULL;
	if(firstnode==NULL)
	{
		firstnode=newnode;
		currentnode=newnode;
	}
	else
	{
```

```
            newnode->prev=currentnode;
            currentnode->next=newnode;
            currentnode=newnode;
        }
}
        void display_forward()
        {
            struct Node*temp;
            temp=firstnode;
            while(temp!=NULL)
            {
                printf("%d\n",temp->data);
                temp=temp->next;
            }
        }
void display_backward()
{
            struct Node*temp;
            temp=currentnode;
            while(temp!=NULL)
            {
                printf("%d\n",temp->data);
                temp=temp->prev;
            }
}
```

```
        void insert_begin()
        {
                int x;
                struct Node*newnode;
                newnode=(struct Node*)malloc(sizeof(struct Node));
                printf("enter the data to insert");
                scanf("%d",&x);
                newnode->prev=NULL;
                newnode->data=x;
                newnode->next=firstnode;
                firstnode->prev=newnode;
                firstnode=newnode;
        }
void insert_end()
{
        int x;
struct Node*newnode;
                newnode=(struct Node*)malloc(sizeof(struct Node));
                printf("enter the data to insert");
                scanf("%d",&x);
                currentnode->next=newnode;
                newnode->prev=currentnode;
                currentnode=newnode;
                newnode->data=x;
                newnode->next=NULL;
```

```
}
int insert_middle_pos()
{
        int ct=0,pos;
        printf("enter the pos");
        scanf("%d",&pos);
        struct Node*temp;
        temp=firstnode;
        while(temp!=NULL)
                {
                if(ct!=pos-1)
                {
                        ct++;
                }
                        else
                        {
                                break;
                        }
                        temp=temp->next;
                }
                struct Node*newnode;
                newnode=(struct Node*)malloc(sizeof(struct Node));
                printf("enter the data");
                scanf("%d",&newnode->data);
                newnode->next=temp->next;
```

```
                    newnode->prev=temp;
                    temp->next=newnode;
                }
        int sum_of_elements()
{
  struct Node *temp;
  temp=firstnode;
  int sum=0;
  while(temp!=NULL)
  {
     sum=sum+temp->data;
     temp=temp->next;
  }
  printf("\n sum of elements is %d\n",sum);
}
void insert_middle_key()
{
        int key;
        printf("enter the value of key");
        scanf("%d",&key);
        int status=0;
        struct Node*temp;
        temp=firstnode;
        while(temp!=NULL)
        {
```

```
            if(temp->data==key)
            {
                status=1;
                break;
            }
            temp=temp->next;
        }
        if(status==1);
        {
            struct Node*newnode;
            newnode=(struct Node*)malloc(sizeof(struct Node));
            printf("enter the data to insert at key position");
            scanf("%d",&newnode->data);
            newnode->next=temp->next;
            newnode->prev=temp;
            temp->next=newnode;
        }
}
int count_of_elements()
{
  struct Node *temp;
  int ct=0;
  temp=firstnode;
  while(temp!=NULL)
  {
```

```
    ct++;
    temp=temp->next;
  }
  return ct;
}
void insert_middle_key_backward()
{
        int key;
        printf("enter the value of key");
        scanf("%d",&key);
        int status=0;
        struct Node*temp;
        temp=firstnode;
        while(temp!=NULL)
        {
                if(temp->data==key)
                {
                        status=1;
                        break;
                }
                temp=temp->next;
        }
        if(status==1);
        {
                struct Node*newnode;
```

```
            newnode=(struct Node*)malloc(sizeof(struct Node));
            printf("enter the data to insert at key position");
            scanf("%d",&newnode->data);
            newnode->next=temp->next;
            newnode->prev=temp;
            temp->next=newnode;
            newnode->prev->next=newnode;
        }
}
void delete_begin()
{
        struct Node*temp;
        temp=firstnode;
        firstnode=firstnode->next;
        firstnode->prev=NULL;
        free(temp);
}
void delete_end()
{
        struct Node*temp,*temp1;
        temp=firstnode;
        while(temp->next!=NULL)
        {
            temp1=temp;
            temp=temp->next;
```

```
        }
    currentnode=temp1;
    temp1->next=NULL;
        free(temp);
}
void DeleteAfter_Key()
{
    int key,find=0;
    printf("\nEnter the key : ");
    scanf("%d",&key);
    struct Node *temp;
    temp=firstnode;
    while(temp!=NULL)
    {
        if(temp->data == key)
        {
            find =1;
            break;
        }
        temp = temp->next;
        }
    if(find==1)
  {
        temp->prev->next=temp->next;
        temp->next->prev=temp->prev;
```

```
        free(temp);
    }
    else
   {
        printf("\n Element not found");
    }

}
        int main()
        {
                insert_create(40);
                insert_create(30);
                insert_create(80);
        insert_begin();
                display_forward();
                insert_end();
                display_backward();
                insert_middle_pos();
                display_forward();
                sum_of_elements();
                printf("%d",count_of_elements());
                insert_middle_key();
            display_forward();
            insert_middle_key_backward();
            display_forward();
```

```
        delete_begin();
        display_forward();
        delete_end();
        display_forward();
        DeleteAfter_Key();
        display_forward();

    }
```

9.Circular Single Linked List operations create and display

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int data;
    struct Node*next;
}*firstnode,*currentnode;
void insert(int x)
{
    struct Node*newnode;
    newnode=(struct Node*)malloc(sizeof(struct Node));
    newnode->data=x;
    newnode->next=NULL;
    if(firstnode==NULL)
    {
        firstnode=newnode;
```

```
            currentnode=newnode;
        }
        else
        {
        currentnode->next=newnode;
        currentnode=newnode;
        }
        currentnode->next=firstnode;
}
void display()
{
        struct Node*temp;
        temp=firstnode;
        if(temp==NULL){
        printf("no elements exist");
}
        do
        {
                printf("%d\n",temp->data);
                temp=temp->next;
        }
        while(temp!=firstnode);
}
void insert_begin()
{
```

```
        int x;
        struct Node*newnode;
        newnode=(struct Node*)malloc(sizeof(struct Node));
        printf("enter the element to insert");
        scanf("%d",&x);
        newnode->data=x;
        newnode->next=firstnode;
        firstnode=newnode;
        currentnode->next=firstnode;
}
void insert_end()
{
        int x;
        struct Node*newnode;
        newnode=(struct Node*)malloc(sizeof(struct Node));
        printf("enter the element to insert at end");
        scanf("%d",&x);
        newnode->data=x;
        currentnode->next=newnode;
        currentnode=newnode;
        currentnode->next=firstnode;
}
void delete_begin()
{
        struct Node*temp;
```

```
        temp=firstnode;
        firstnode=firstnode->next;
        currentnode->next=firstnode;
        free(temp);
}
void delete_end()
{
        struct Node*temp,*temp1;
        temp=firstnode;
        while(temp->next!=firstnode)
        {
                temp1=temp;
                temp=temp->next;
        }
     currentnode=temp1;
         temp1->next=NULL;
         free(temp);
}
void main()
{
        insert(10);
        insert(20);
        insert(30);
        display();
        insert_begin();
```

```
        display();
        insert_end();
        display();
        delete_begin();
        display();
        delete_end();
        display();
}
```

10. Stack with Linked List implementation.

```
#include<stdio.h>
#include<stdlib.h>
struct stack
{
        int data;
        struct stack*next;
}*top=NULL;
void push(int data)
{
        struct stack*newnode;
        newnode=(struct stack*)malloc(sizeof(struct stack));
        newnode->data=data;
        newnode->next=top;
        top=newnode;
}
void pop()
```

```
{
        struct stack*temp;
        temp=top;
        printf("enter the deleted element %d ",temp->data);
        top=top->next;
        free(temp);
}
void display()
{
        struct stack*temp;
        temp=top;
        while(temp!=NULL)
        {
                printf("%d\n",temp->data);
                temp=temp->next;
        }
}
void peep()
{
        printf("%d\n",top->data);
}
void isempty()
{
        if(top==NULL)
        {
```

```
            printf("stack is empty");
        }
}
void main()
{
int option =1,l;
int choice;
   while (option)
   {
    printf ("----------------------------------------\n");
    printf ("       1    -->    push              \n");
    printf ("       2   -->    display          \n");
    printf ("      3   -->    pop          \n");
    printf ("      4  -->    peep         \n");
    printf("           5-->   is empty    \n");
    printf ("----------------------------------------\n");

    printf ("Enter your choice\n");
      scanf    ("%d", &choice);
      switch (choice)
      {
      case 1:
      printf("enter element");
          scanf("%i",&l);
          push(l);
```

```
    break;
case 2:
    printf("enter element");
    scanf("%i",&l);
    push(l);

    break;
case 3:
    printf("enter element");
    scanf("%i",&l);
    push(l);
    break;
case 4:
display();
break;
case 5:
 pop();
 break;
 case 6:
 display();
 break;
 case 7:
 peep();
 break;
```

```
        case 8:
        isempty();
        break;
        default:
        printf("\nwrong choice");
        }

       printf("\nDo you want to continue(Type 0 or 1)?\n");
       scanf("%d", &option);
    }
}
```

11. Queue with Linked List implementation.

```
#include<stdio.h>
#include<stdlib.h>
struct queue
{
  int data;
  struct queue*next;
}*front=NULL,*rear=NULL;
void enqueue(int x)
{
  struct queue*newnode;
  newnode=(struct queue*)malloc(sizeof(struct queue));
  newnode->data=x;
  newnode->next=NULL;
```

```
  if(front==NULL)
  {
    rear=front=newnode;
  }
   else
  {
    rear->next=newnode;
    rear=newnode;
  }
}
void display()
{
  struct queue*temp;
  temp=front;
  while(temp!=NULL)
  {
    printf("%d\n",temp->data);
    temp=temp->next;
  }
}
void dequeue()
{
  struct queue*temp;
  temp=front;
  printf("the deleted element%d\n",front->data);
```

```
  front=front->next;
  free(temp);
}
void Isempty()
{
  if(front==NULL)
  {
       printf("empty");
  }
}
void main()
{
int option=1,x;
int choice;
    while (option)
    {
       printf ("1-->enqueue\n");
       printf ("2-->display\n");
       printf ("3-->dequeue\n");
    printf ("4-->Isempty\n");
       printf ("Enter your choice\n");
       scanf("%d", &choice);
       switch(choice)
       {
         case 1:
```

```
            printf("enter element\n");
            scanf("%d",&x);
            enqueue(x);
            break;
        case 2:
            display();
            break;
        case 3:
            dequeue();
            break;
        case 4:
            Isempty();
      break;
      default:
        printf("\nwrong choice");
        }
        printf("\nDo you want to continue(Type 0 or 1)?\n");
        scanf("%d", &option);
    }
}
```

12. Circular queue with array implementation

```
#include<stdio.h>
#include<stdlib.h>
#define size 10
int front=-1;
```

```
int rear=-1;
int queue[size];
void insert(int ele)
{

        if(front==-1 && rear==-1)
        {
                front=rear=0;
                queue[rear]=ele;
        }
        else if((rear+1)%size==front)
        printf("queue is full");
        else
        {
                rear=(rear+1)%size;
                queue[rear]=ele;
        }
}
void delete()
{
  int front,rear;
  if(front==-1&&rear==-1)
  printf("queue is empty");
  else {
  if(front==rear)
```

```
  {
      printf("the delete ele %d",queue[front]);
      front =rear=-1;
        }
        else
        {
            printf("the delete ele %d",queue[front]);
            front=(front+1)%size;
        }
}
}
void display()
{
      int i=front;
      if(front<=rear)
      {
            for(i=front;i<=rear;i++)
                  printf("%d",queue[i]);
            }
      else
      {
            for(i=front;i<size;i++)
                  printf("%d",queue[i]);
            {
            for(i=0;i<=rear;i++)
```

```
                printf("%d",queue[i]);
            }
        }
}
void main()
{
        int option =1,l;
int choice;
   while (option)
   {
    printf ("----------------------------------------\n");
    printf ("       1    -->    insert             \n");
    printf ("       2   -->    delete          \n");
    printf ("      3   -->    display         \n");
    printf ("----------------------------------------\n");

    printf ("Enter your choice\n");
      scanf    ("%d", &choice);
      switch (choice)
      {
      case 1:
      printf("enter element");
          scanf("%i",&l);
          insert(l);
```

```
            break;
        case 2:
            printf("enter element");
            scanf("%i",&l);
            delete(l);

            break;
        case 3:
            display();
            break;
         default:
         printf("\nwrong choice");
        }

        printf("\nDo you want to continue(Type 0 or 1)?\n");
        scanf("%d", &option);
    }
}
```

13. Double ended Queue operations (enQueueFront , enQueueFront, deQueueRear , deQueueRear)

Tracing

```
#include<stdio.h>
#include<stdlib.h>
#define size 10
```

```
int front=-1,rear=-1;
int dequeue[size];
void insert_at_front(int x)
{
	if((front==0 && rear==size-1)||(front==rear+1))
	{
		printf("queue overflow");
	}
	else if(front==-1 && rear==-1)
	{
		front=rear=0;
		dequeue[front]=x;
	}
	else if(front==0)
	{
		front=size-1;
		dequeue[front]=x;
	}
	else
	{
		front=front-1;
		dequeue[front]=x;
	}
}
void delete_at_rear()
```

```
{
        if(front==-1 && rear==-1)
        {
                printf("queue is empty");
        }
        else if(front==rear)
        {
                printf("the element to be deleted at rear end is
%d",dequeue[rear]);
                front=rear=-1;
        }
        else if(rear==0)
        {
                printf("the element to be deleted at rear end is
%d",dequeue[rear]);
                rear=size-1;
        }
        else
        {
        printf("the element to be deleted at rear end is
%d",dequeue[rear]);
        rear=rear-1;
        }
}
void insert_at_rear(int x)
{
```

```
        if((front==0 && rear==size-1)||(front==rear+1))
        {
                printf("queue is full");
        }
                if(front==-1 && rear==-1)
                {
                        front=rear=0;
                        dequeue[rear]=x;
                }
                else
                {
                        rear=rear+1;
                        dequeue[rear]=x;
                }
}
void delete_at_front()
{
        if(front==-1 && rear==-1)
        {
                printf("queue is empty");
        }
        else if(front==rear)
        {
                printf("the element deleted is %d",dequeue[front]);
                front=rear=-1;
```

```
        }
        else if(front==size-1)
        {
                printf("the element deleted is %d",dequeue[front]);
                front=0;
        }
        else
        {
                printf("\nthe element deleted is %d",dequeue[front]);
                front=front+1;
        }
}
void display()
{
        int i=front;
        printf("\nthe elements in the queue are");
        while(i!=rear)
        {
                printf("%d",dequeue[i]);
                i=(i+1)%size;
        }
        printf("%d ",dequeue[i]);

}
void main()
```

```
{
insert_at_front(10)     ;
delete_at_rear();
insert_at_rear(20);
insert_at_rear(30);
delete_at_front();
display();
}
```

14. program on BST insert ,delete , traversal

```
#include<stdio.h>
#include<stdlib.h>
struct BST
{
        struct BST*left;
        int data;
        struct BST*right;
}*root=NULL;
void insert(struct BST*temp,struct BST*newnode);
void create()
{
        struct BST*newnode;
        newnode=(struct BST*)malloc(sizeof(struct BST));
        newnode->left=NULL;
        newnode->right=NULL;
        printf("enter the data element");
```

```
        scanf("%d",&newnode->data);
        if(root==NULL)
        root=newnode;
        else
        insert(root,newnode);
}
void insert(struct BST*temp,struct BST*newnode)
{
        if(newnode->data<temp->data)
        {
           if(temp->left==NULL)
           temp->left=newnode;
           else
           insert(temp->left,newnode);

        }
        if(newnode->data>temp->data)
        {
           if(temp->right==NULL)
           temp->right=newnode;
           else
           insert(temp->right,newnode);
        }
}
        void inorder(struct BST*temp)
```

```
{
        if(temp!=NULL)
        {
                inorder(temp->left);
                printf("%d ",temp->data);
                inorder(temp->right);

        }
}
void preorder(struct BST*temp)
{
        if(temp!=NULL)
        {
                        printf("%d ",temp->data);
                preorder(temp->left);
                preorder(temp->right);
        }
}void postorder(struct BST*temp)
{
        if(temp!=NULL)
        {
                postorder(temp->left);
                postorder(temp->right);
                printf("%d ",temp->data);
        }
```

```
        }
int findmin(struct BST*temp)
{
        while(temp->left!=NULL)
        {
                temp=temp->left;
        }
        printf("%d",temp->data);
}
int findmax(struct BST*temp)
{
        while(temp->right!=NULL)
        {
                temp=temp->right;
        }
        printf("%d",temp->data);
}
int search(struct BST*temp,int ele)
{
        if(temp==NULL)
        return 0;
        else if(temp->data==ele)
        return 1;
        else if(ele<temp->data)
        search(temp->left,ele);
```

```
        else
        search(temp->right,ele);
}
struct BST*findmin_sucr(struct BST*temp)
{
        while(temp->left!=NULL)
        {
                temp=temp->left;
        }
        return temp;
}
struct BST*findmax_pred(struct BST*temp)
{
  while(temp->right!=NULL)
  {
    temp=temp->right;
  }
  return temp;
}
void delete_node(struct BST*root,int key)
{
        struct BST*parent,*current;
        parent=NULL;
        current=root;
        //search for key in BST
```

```
while(current!=NULL &&current->data!=key)
{
        parent=current;
        if(key<current->data)
        {
        current=current->left;
}
        else
        {
        current=current->right;
}
//if key not found
        if(current==NULL)
        return;
}
//case 1:node having 0 children

        if(current->left==NULL && current->right==NULL)
        {
                if(current!=root)
{
        if(parent->left==current)
        parent->left=NULL;
        else
        parent->right=NULL;
```

```
            }
            else
            {
                root=NULL;
            }
        }
//case 2:having 2 child
  else if(current->left!=NULL && current!=NULL)
  {
        struct BST*sucr;
        sucr=findmin_sucr(current->right);
        int val=sucr->data;
        delete_node(root,sucr->data);
        current->data=val;
          }

          //case 3:node with 1 child
          else
          {
                struct BST*child;
                child=(current->left!=NULL)?current->left:current->right;
                if(current!=root)
                {

                if(parent->left==current)
```

```
                {
                        parent->left=child;
                }
                        else
                        {
                        parent->right=child;
                 }
        }
                 else
                 {
                        root=child;
                 }
}
}

void main()
{
        int nodes,i,key;
        printf("enter no of nodes\n");
        scanf("%d",&nodes);
        for(i=0;i<nodes;i++)
        create();
        printf("inorder is\n");
        inorder(root);
                printf("\npreorder is\n");
```

```
        preorder(root);
                printf("\npostorder is\n");
        postorder(root);
                printf("\nmin is\n");
        findmin(root);
                printf("\nmax is\n");
        findmax(root);
        printf("enter the key to search");
        scanf("%d",&key);
        if(search(root,key))
        printf("element is found");
        else
        printf("element is not found");
        printf("enter the element to be deleted");
        scanf("%d",&key);
        delete_node(root,key);
        inorder(root);
}
```

15. program on heap sort

```
#include<stdio.h>
void main()
{
  int arr[20],n,i;
  printf("enter the number of ele");
  scanf("%d",&n);
```

```
  for( i=0;i<n;i++)
  {
    scanf("%d",&arr[i]);
  }
  heapsort(arr,n);
  printf("The elements after sorting are :");
  for(i=0;i<n;i++)
  {
    printf(" %d",arr[i]);
  }
}
void heapsort(int arr[],int n)
{
  //construct max heap//
  int i;
  for( i=(n/2-1);i>=0;i--)
  heapify(arr,n,i);
  //sorting of elements by deleting the root node//
  for( i=n-1;i>=0;i--)
  {
    swap(&arr[0],&arr[i]);
    heapify(arr,i,0);
  }

}
```

```
void heapify(int arr[],int n,int i)
{
  int large = i;
  int left=2*i+1 ,right = 2*i+2;
  if(left<n && arr[left]>arr[large])
  {
    large=left;
  }
  if(right<n && arr[right]>arr[large])
  {
    large=right;
  }
  //if large isnt a root continue heapify//
  if(large!=i)
  {
    swap(&arr[i],&arr[large]);
    heapify(arr,n,large);
  }
}
void swap(int *a,int *b)
{
  int temp;
  temp = *a;
  *a=*b;
  *b=temp;
```

}