Exp.No:2.1(a) Date :	**ABSTRACT DATA TYPES OF QUEUE**

AIM:

To write a C-program to Perform Enqueue(), Dequeue(),Display()operations on the given data structure.

PSEUDOCODE:

```
BEGIN
    Define a size variable
    Declare front,rear,q[size]
    void enqueue(int ele)
        if(rear==max-1)
        print"QueueOverflow"
        elseif((rear==-1)&&(front==-1))
        front=rear=0;
        else
        rear=rear+1;
    queue[rear]=ele;
    void dequeue()
        if(front==-1)
        print"QueueUnerflow"else
        print"DequeuedElement",queue[front]
        if(front==rear)
        front=rear=-1;
        else
        front++;
    void display()
        if((rear==-1)||(front==-1))
        print "Queue Empty "else
        print "Elements:
        "for(i=front;i<=rear;i++)
        print " queue[i] "
END
```

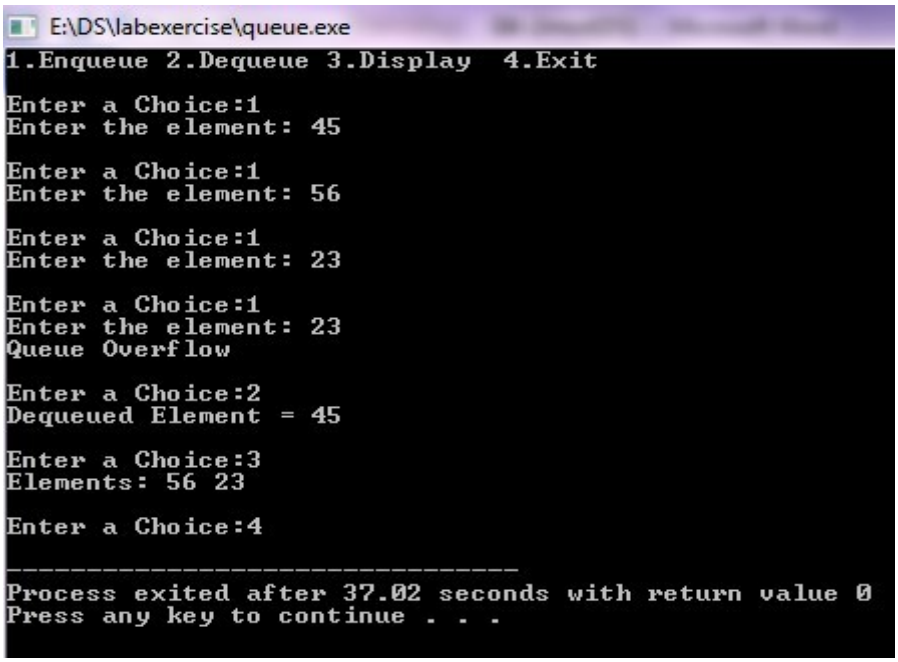
SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
#define max 3
int queue[max];
int front=-1,rear=-1;
voidenqueue(intele){
if(rear==max-1){
```

```
        printf("QueueOverflow\n");
        return;
}
elseif((rear==-1)&&(front==-1))
        front=rear=0;
else
        rear=rear+1;
queue[rear]=ele;
}
voiddequeue(){
if(front==-1)
        printf("QueueUnerflow\n");
else{
        printf("DequeuedElement=%d\n",queue[front]); if(front==rear)
                front=rear=-1;
        else
                front++;
}}
voiddisplay(){
int i;
if((rear==-1)||(front==-1))
        printf("QueueEmpty\n");
else{
        printf("Elements: ");
        for(i=front;i<=rear;i++)
                printf("%d",queue[i]);
        printf("\n");
}}
void main(){
int choice,ele;
printf("1.Enqueue2.Dequeue3.Display4.Exit\n");
while(1){
        printf("\nEnteraChoice:");
        scanf("%d",&choice);
        switch(choice){
                case 1:
                        printf("Entertheelement:");
                        scanf("%d",&ele);
                        enqueue(ele);
                        break;
                case 2:
                        dequeue();
                        break;
                case 3:
                        display();
                        break;
                case 4:
                        exit(0);
                default:
```

```
                printf("invalid");
            }
        }}
```

OUTPUT:



RESULT:

Thus, the C program to Perform Enqueue(), Dequeue(), Display() operations is Executed and the output is verified successfully.

Exp.No:2.1(b) Date :	**ABSTRACT DATA TYPES OF CIRCULAR QUEUE**

AIM:

To write a C-program to implement the circular queue data structure.

PSEUDOCODE:

```
BEGIN
    Define a size variable
    Declarefront,rear,q[size]
    void enqueue(int ele)
        if((rear==max-1)&&(front==0))
        printf("Queue Overflow\n");
        elseif((rear==-1)&&(front==-1))
        front=rear=0;
        elseif((rear==max-1)&&(front!=0))
        rear=0;
        else
        rear=rear+1;
    queue[rear]=ele;
    void dequeue()
        if(front==-1)
        print"QueueUnerflow"else
        print"DequeuedElement=",queue[front]
        if(front==rear)
        front=rear=-1;
        elseif(front==max-1)
        front=0;
        else
        front++;
    void display()
        if((rear==-1)||(front==-1))
        print "Queue Empty"
        else if(rear>=front)
        for(i=front;i<=rear;i++)
        print " queue[i]"
        else
        for(i=front;i<max;i++)
        print
        "queue[i]"for(i=0;i<=re
        ar;i++) print " queue[i]"
END
```

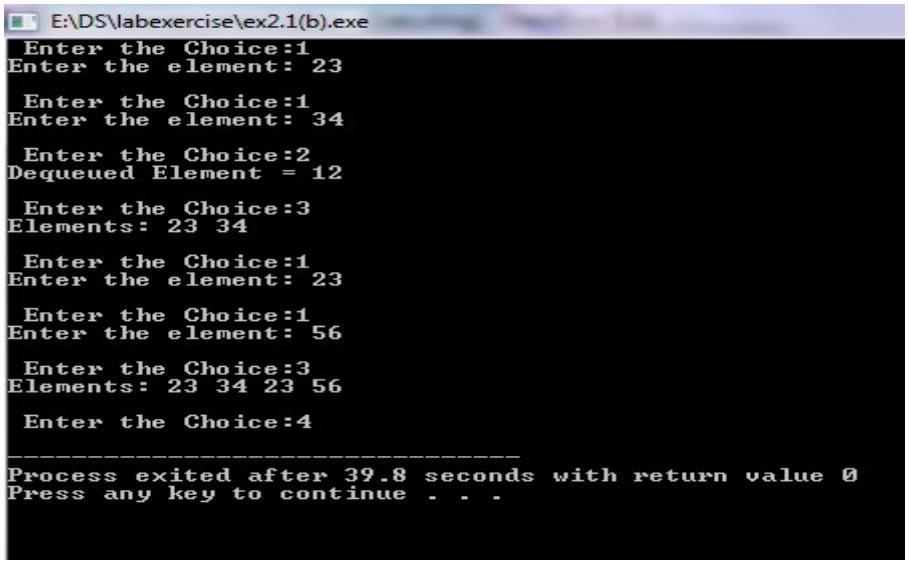
SOURCE CODE:

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

```
#include<stdlib.h>
#define max 4
int queue[max];
int front=-1,rear=-1;
voidenqueue(intele){
if((rear==max-1)&&(front==0))
        printf("QueueOverflow\n");
elseif((rear==-1)&&(front==-1))
        front=rear=0;
elseif((rear==max-1)&&(front!=0))
        rear=0;
else
        rear=rear+1;
queue[rear]=ele;
}
voiddequeue(){
if(front==-1)
        printf("QueueUnerflow\n");
else{
        printf("DequeuedElement=%d\n",queue[front]); if(front==rear)
                front=rear=-1;
        elseif(front==max-1)
                front=0;
        else
                front++;
}}
voiddisplay(){
int i;
if((rear==-1)||(front==-1))
        printf("QueueEmpty\n");
elseif(rear>=front){
        printf("Elements: ");
        for(i=front;i<=rear;i++)
                printf("%d",queue[i]);
        printf("\n");
}
else{
        printf("Elements: ");
        for(i=front;i<max;i++)
                printf("%d",queue[i]);
        for(i=0;i<=rear;i++)
                printf("%d",queue[i]);
        printf("\n");
}}
int
main(){intch
oice,ele;
printf("1.Enqueue2.Dequeue3.Display4.Exit\n");
while(1){
        printf("\nEnterthe Choice:");
```

```
            scanf("%d",&choice);
            switch(choice){
                  case1:printf("Entertheelement:");
                              scanf("%d",&ele);
                              enqueue(ele);
                              break;
                  case2:dequeue();
                              break;
                  case 3: display();
                              break;
                  case 4:
                              exit(0);
                  default:
                              printf("invalid");
            }}
      return0;
      }
```

OUTPUT:



RESULT:

Thus the C program implement the circular queue data structure is executed and the output is verified successfully.

.

Exp.No:2.2(a) Date :	**DOUBLE ENDED QUEUE**

AIM:

To write a C-program to implement the Double Ended Queue data structure.

Data Structure used:
Loop Data Type: integer
Routine: Iterative

PSEUDOCODE:

```
BEGIN
                    Void enqueue_front(intx){
              if((f==0 && r==size-1)||f==r+1){
              printf("Overflow");
                    return;
                    }elseif(f==-1&&r==-1)
                    f=r=0;
              else
                    r++;
              g[r]=x;}
        void enqueue_rear(int
              x){if(r==size-1){
                    printf("Overflow")
                    ;return;}
                elseif(f==-1&&r==-1)
                    f=r=0;
              else
                    r++;
              g[r]=x;}
        void dequeue_rear(){
              if(r==size-1){
                    printf("Underflow");
                    return;}
              elseif(f==r)
                    f=r=-1;
              else
                    r--;}
        void dequeue_front(){
              if(f==-1){
                    printf("Underflow");
                    return;}
              elseif(f==r)
                    f=r=-1;
              else
                    f+=1;}
END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
#define size 10
 int
  g[size];intf=-
  1,r=-1;
 void enqueue_front(int x){
if((f==0&&r==size-1)||f==r+1){
  printf("Overflow");
 return;}
  elseif(f==-1&&r==-1)
  f=r=0;
   else
         r++;
   g[r]=x;}
voidenqueue_rear(intx){
  if(r==size-1){
    printf("Overflow");
    return;}
 elseif(f==-1&&r==-1) f=r=0;
 else
    r++;
    g[r]=x;}
void dequeue_rear(){
 if(r==size-1){
 printf("Underflow");
   return;}
 elseif(f==r)
  f=r=-1;
 else
 r--;}
  voiddequeue_front(){
 if(f==-1){
   printf("Underflow");
  return;}
  elseif(f==r)
  f=r=-1;
    else
     f+=1;}
 voiddisplay(){
   if(f==-1){
 printf("Underflow");
      return;
  }
    else{
       int i;
     for(i=f;i<=r;i++){
```

```
            printf("%d",g[i]);}
               printf("\n");}}
              int main(){
                 intch,val;
                 while(1){
             printf("1.EnqueueAtFront
             2.EnqueueAtRear
             3.DequeueAtFront
             4.DequeueAtRear
             5.Display
             6.Exit\n");

               printf("Enterchoice:");
                 scanf("%d",&ch);
                 switch(ch){

  case 1:
          printf("Entervalue:");
          scanf("%d",&val);
          enqueue_front(val);
          break;

  case 2:
       printf("Entervalue:");
                 scanf("%d",&val);
                 enqueue_rear(val);
                 break;

       case 3:
                 dequeue_front();
                 break;

       case4:
                 dequeue_rear();
                 break;

       case5:
                 display();
                 break;

       case 6:
                 exit(0);

       default:
                 printf("InvalidInput");break;

    }
  }
}
```

OUTPUT:

```
Enter choice:1
Enter value:30
1.EnqueueAtFront 2.EnqueueAtRear 3.DequeueAtFront 4.DequeueAtRear 5.Display 6.Exit
Enter choice:1
Enter value:40
1.EnqueueAtFront 2.EnqueueAtRear 3.DequeueAtFront 4.DequeueAtRear 5.Display 6.Exit
Enter choice:5
10 20 30 40
1.EnqueueAtFront 2.EnqueueAtRear 3.DequeueAtFront 4.DequeueAtRear 5.Display 6.Exit
Enter choice:2
Enter value:50
1.EnqueueAtFront 2.EnqueueAtRear 3.DequeueAtFront 4.DequeueAtRear 5.Display 6.Exit
Enter choice:5
10 20 30 40 50
1.EnqueueAtFront 2.EnqueueAtRear 3.DequeueAtFront 4.DequeueAtRear 5.Display 6.Exit
Enter choice:3
1.EnqueueAtFront 2.EnqueueAtRear 3.DequeueAtFront 4.DequeueAtRear 5.Display 6.Exit
Enter choice:5
20 30 40 50
1.EnqueueAtFront 2.EnqueueAtRear 3.DequeueAtFront 4.DequeueAtRear 5.Display 6.Exit
Enter choice:4
1.EnqueueAtFront 2.EnqueueAtRear 3.DequeueAtFront 4.DequeueAtRear 5.Display 6.Exit
Enter choice:5
20 30 40
1.EnqueueAtFront 2.EnqueueAtRear 3.DequeueAtFront 4.DequeueAtRear 5.Display 6.Exit
Enter choice:6

--------------------------------
Process exited after 59.58 seconds with return value 0
Press any key to continue . . .
```

RESULT:

Thus the C program implement the circular queue data structure is executed and the output is verified successfully.

.

Exp.No :2.2(b) Date:	**RANDOM STRING**

AIM:

To write a C-program to implement the circular queue data structure.

Data Structure used: Loop

Data Type: integer

Routine: Iterative

PSEUDOCODE:

-

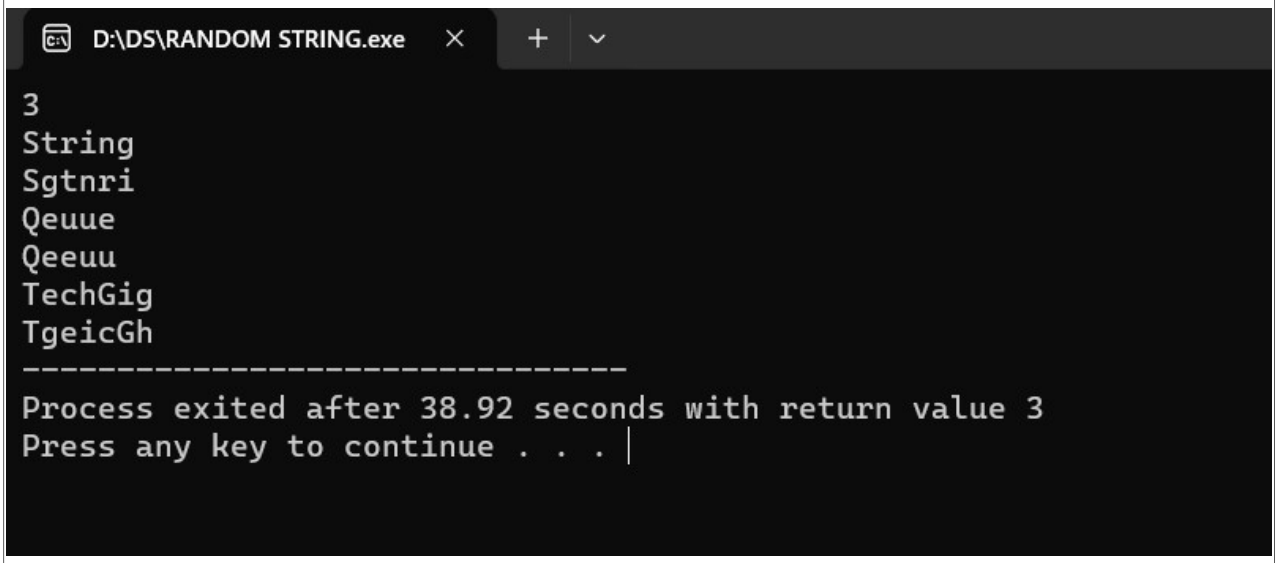
```
BEGIN
        Void enqueue(charx,int y){
          if(r==y-1){
                        printf("Overflow");
                        return;}
                elseif(f==-1&&r==-1)
                        f=r=0;
          else
                        r=r+1;
          s[r]=x;}
      void dequeue(){
          if(f==-1){
                 printf("Underflow");
                        return;}
          elseif(f==r)
                        f=r=-1;
          else
                        f+=1;}
      void display(){
          if(f==-1)
          {
                 printf("Underflow");
                        return;}
          else{
                        int i,j;
                        if((r+1)%2==0){
                          for(i=f,j=r;i<=j;i++,j--){
                                printf("%c",s[i]);
                                printf("%c",s[j]);}
                         for(i=0;i<=r;i++)
                                dequeue();}
                        else{
                         for(i=f,j=r;i<j;i++,j--){
                                printf("%c",s[i]);
                                printf("%c",s[j]);}
                               if(i==j)
                                printf("%c",s[i]);
                         for(i=0;i<=r;i++)
                                dequeue();}}}
```

SOURCE CODE:

```
            #include<stdio.h>
#include<stdlib.h>
#include<string.h>
   int f=-1,r=-1;
      chars[100];
   voidenqueue(charx,inty){
     if(r==y-1){
        printf("Overflow");
   return;
   }
 elseif(f==-1&&r==-1) f=r=0;
else
  r=r+1;
   s[r]=x;}
  void dequeue(){
 if(f==-1){
 printf("Underflow");
 return;}
 elseif(f==r)
 f=r=-1;
 else
f+=1;}
void display(){
 if(f==-1){
 printf("Underflow");
 return;
}
   else{
    int i,j; if((r+1)%2==0){
  for(i=f,j=r;i<=j;i++,j--){
   printf("%c",s[i]);
printf("%c",s[j]);}
        for(i=0;i<=r;i++)
 dequeue();}
else{for(i=f,j=r;i<j;i++,j--){
  printf("%c",s[i]);
   printf("%c",s[j]);}
    if(i==j)
    printf("%c",s[i]);
 for(i=0;i<=r;i++)
 dequeue();}}}int
 main(){
  int n,i,j;
 scanf("%d",&n);
 for(i=0;i<n;i++){
     scanf("%s",s);
```

```
        int len=strlen(s);
    for(j=0;s[j]!='\0';j++)
    enqueue(s[j],len);
      display();}}
```

OUTPUT:



RESULT:

Thus, the C program implement the circular queue data structure is executed and the output is verified successfully.

Exp.No:2.3(a) Date :	**SINGLY LINKED LIST**

AIM:

To write a C-program to implement the circular queue data structure.
Data Structure used: Loop
Data Type: integer
Routine: Iterative

PSEUDOCODE:

```
BEGIN
    structnode{
            int data;
            struct node*next;};
      struct
      node*head=NULL,*tail=NULL;void
      insert_end(int val){
            structnode*n=(structnode*)malloc(sizeof(struct
            node));n->data=val;
            n->next=NULL;
            if(head==NULL
            )
                  head=tail=n;
            else{
                  tail-
                  >next=n;
                  tail=n;}}
      voidinsert_begin(int val){
            structnode*n=(structnode*)malloc(sizeof(struct
            node));n->data=val;
            n->next=NULL;
            if(head==NULL
            )
                  head=tail=n;
            else{
                  n-
                  >next=head;
                  n=head;}}
      voidinsert_middle(intval,int pos){
            structnode*n=(structnode*)malloc(sizeof(struct
            node));n->data=val;
            n-
            >next=NULL;
            struct      node
            *i,*t;int k=0;
            for(i=head;k<pos-1;i=i-
                  >next){t=i->next;
```

```
                    i-
                    >next=n;
                    n-
                    >next=t;
                    k++;}}
          void delete_begin(){
                if(head==NULL){
                    printf("Underflow");
                    return;}
                else{
                    structnode*t;
           t=head;
           head=head->next;
           free(t);}}
   void delete_end(){
     structnode*i,*t;
     t=tail;
     for(i=head;i->next->next!=NULL;i=i->next);
     i->next=NULL;
     free(t);}
voiddelete_middle(intpos){
   struct node *t,*i;
     int k=0;
     for(i=head;k<pos-1;i=i->next){
          t=i->next;
           i->next=t->next;
           free(t);
           k++;}}
void display(){
     structnode*i;
     for(i=head;i!=NULL;i=i->next)
     printf("%d ",i->     data);
     printf("\n");}

   END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>st
ruct node{
  int data;
    struct node* next;};
  structnode*head=NULL,*tail=NULL;
  void insert_end(int val){
 structnode*n=(structnode*)malloc(sizeof(structnode));
   n->data=val;
    n->next=NULL;
     if(head==NULL)
     head=tail=n;
```

```
                    else{
                    tail->next=n;
                   tail=n;}}
                voidinsert_begin(intval){
               structnode*n=(structnode*)malloc(sizeof(structnode));
                 n->data=val;
                  n->next=NULL;
                  if(head==NULL)
                head=tail=n;
                else{
               n->next=head;
               n=head;}}
             voidinsert_middle(intval,intpos){
            structnode*n=(structnode*)malloc(sizeof(structnode));
            n->data=val;
             n->next=NULL;structnode*i,*t;intk=0;
            for(i=head;k<pos-1;i=i->next){ t=i->next;
            i->next=n; n->next=t; k++;}}
            voiddelete_begin(){
           if(head==NULL){
            printf("Underflow");return;}
            else{
          structnode*t;t=head;head=head->next;free(t);}} void
          delete_end(){
         structnode*i,*t; t=tail;
          for(i=head;i->next->next!=NULL;i=i->next);i->next=NULL;
           free(t);}
      voiddelete_middle(intpos){structnode*t,*i; int
          k=0;
         for(i=head;k<pos-1;i=i->next){t=i->next;
           i->next=t->next; free(t);
            k++;}}
      void display(){
       structnode*i;for(i=head;i!=NULL;i=i->next) printf("%d
         ",i->data);
          printf("\n");}
       int main(){
       intval,i,ch,n;
       while(1){
          printf("1.InsertAtBegin2.InsertAtEnd3.InsertAtMiddle4.DeleteAtBegin
5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit\n");
        printf("EnterChoice:");
         scanf("%d",&ch);
            switch(ch){
    case 1:
          printf("Entervalue:");
          scanf("%d",&val);
          insert_begin(val);
          break;
```

```
  case 2:
                printf("Enter value:"); scanf("%d",&val); insert_end(val); break;
      case 3:
                printf("Enter value:"); scanf("%d",&val); printf("Enter position:"); scanf("%d",&n);
insert_middle(val,n);
              break;
      case 4:
                delete_begin();
                break;
      case 5:
               delete_end();
               break;
       case 6:
               printf("Enter position");
               scanf("%d",&n);
               delete_middle(n);
               break;
       case 7:
                display();
                break;
       case 8:
                exit(0);
       default:
                printf("INVALID INPUT\n");
                break;
         }
    }
}
```

OUTPUT:

```
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:1
Enter value:10
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:2
Enter value:30
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:7
10 30
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:3
Enter value:20
Enter position:2
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:7
10 20 30
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:2
Enter value:40
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:7
10 20 30 40
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:4
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:7
20 30 40
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:5
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:7
20 30
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:6
Enter position 2
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:7
20
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:8
```

RESULT:

Thus the C program implement the circular queue data structure is executed and the output is verified successfully.

Exp.No:2.3(b) Date :	**DOUBLY LINKED LIST**

AIM:

To write a C-program to implement the circular queue data structure.
Data Structure used: Loop
Data Type: integer
Routine: Iterative

PSEUDOCODE:

```
BEGIN
        voidinsert_end(intval){
        structnode*n=(structnode*)malloc(sizeof(structnode));
        n->data=val;
        n->next=NULL;
        if(head==NULL)
        head=tail=n;
        else{
        tail->next=n;
        tail=n;}}
        voidinsert_begin(intval){
        struct node *i;
        structnode*n=(structnode*)malloc(sizeof(structnode));
        n->data=val;
        n->next=NULL;
        if(head==NULL)
        head=tail=n;
        else{
        i->prev=n;
        n->next=i;
        head=n;}}
        voidinsert_middle(intval,intpos){
        structnode*n=(structnode*)malloc(sizeof(structnode));
        n->data=val;
        n->next=NULL;
        structnode*i,*t;
        int k=0;
        for(i=head;k<pos-1;i=i->next){
        t=i->next;
        i->next=n;n
        ->prev=i;
        n->next=t;
        t->prev=n;
        k++;}}
        voiddelete_begin(){
        if(head==NULL){
        printf("Underflow");
```

```
return;}
else{
structnode* t;
t=head;head=head->next;
head->prev=NULL;
free(t);}}
voiddelete_end(){
struct node *t,*i;
for(i=head;i->next->next!=NULL;i=i->next);
t=tail;
tail=tail->prev;
tail->next=NULL;
free(t);}
voiddelete_middle(intpos){
struct node *i,*t;
int k=0;
for(i=head;k<pos-1;i=i->next){
t=i->next->next;
free(i->next);
i->next=t;
t->prev=i;
k++;}}
```

END

SOURCE CODE:

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

structnode*head=NULL,*tail=NULL;
void insert_end(int val){
 structnode*n=(structnode*)malloc(sizeof(structnode));
n->data=val;
n->next=NULL;
if(head==NULL)
head=tail=n;
else{
tail->next=n;
tail=n;}}
voidinsert_begin(intval){
struct node *i;
structnode*n=(structnode*)malloc(sizeof(structnode));
n->data=val;
n->next=NULL;
```

```
                    if(head==NULL)
                    head=tail=n;
                    else{
                   i->prev=n;
                   n->next=i;
                   head=n;}}
                   voidinsert_middle(intval,intpos){
                    structnode*n=(structnode*)malloc(sizeof(structnode));
                    n->data=val;
                    n->next=NULL;
                    structnode*i,*t;
                    int k=0;
                    for(i=head;k<pos-1;i=i->next){
                    t=i->next;
                    I->next=n;
                    n->prev=i;
                    n->next=t;
                    t->prev=n;
                    k++;}}
               voiddelete_begin(){
               if(head==NULL){
               printf("Underflow");
               return;}
               else{
               struct node* t;
               t=head;
               head=head->next;
               head->prev=NULL;
               free(t);}}
               voiddelete_end(){
               struct node *t,*i;
               for(i=head;i->next->next!=NULL;i=i->next);
               t=tail;
                 tail=tail->prev;
                 tail->next=NULL;
                 free(t);}
          voiddelete_middle(intpos){structnode*i,*t; int
                 k=0;
                 for(i=head;k<pos-1;i=i->next){
                 t=i->next->next;
                 free(i->next);
                 i->next=t;
                 t->prev=i;
                 k++;}}
                 voiddisplay(){
                 struct node *i;
                 for(i=head;i!=NULL;i=i->next){
                 printf("%d ",i->data);}
                 printf("\n");}
                 int main(){
```

```
                    intch,n,val;
                    while(1){
                    printf("1.InsertAtBegin2.InsertAtEnd3.InsertAtMiddle4.DeleteAtBegin
5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit\n");
                    printf("EnterChoice:");
                    scanf("%d",&ch);
                    switch(ch){
            case1:
                    printf("Entervalue:");
                    scanf("%d",&val);
                    insert_begin(val);
                    break;
            case2:
                    printf("Entervalue:");
                    scanf("%d",&val);
                    insert_end(val);break;
            case 3:
                    printf("Enter value:");
                    scanf("%d",&val);
                    printf("Enterposition:");
                    scanf("%d",&n);insert_middle(val,n);
                    break;
            case 4:
                    delete_begin();
                    break;
            case 5:
                    delete_end();
                    break;
            case 6:
                    printf("Enterposition");
                    scanf("%d",&n);
                    delete_middle(n);break;
            case7:
                    display();
                    break;
            case8:
                    exit(0);
                    default:
                    printf("INVALIDINPUT\n");
                    break;
        }
    }
}
```

OUTPUT:

```
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:1
Enter value:10
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:2
Enter value:30
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:3
Enter value:20
Enter position:2
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:7
10 20 30
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:2
Enter value:40
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:7
10 20 30 40
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:4
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:7
20 30 40
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:6
Enter position 2
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:7
20 40
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:5
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:7
20
1.InsertAtBegin 2.InsertAtEnd 3.InsertAtMiddle 4.DeleteAtBegin 5.DeleteAtEnd 6.DeleteAtMiddle 7.Display 8.Exit
Enter Choice:8
```

RESULT:

Thus the C program implement the circular queue data structure is executed and the output is verified successfully.

Exp.No:2.3(c) Date :	**SUM OF SINGLY LINKED LIST**

AIM:

To write a C-program print the sum of singly linked list.

Data Structure used: Loop

Data Type: integer

Routine: Iterative

PSEUDOCODE:

```
BEGIN
    Void insert_end(int val){
        Struct node*n=(struct
        node*)malloc(sizeof(struct node));n->data=val;
        n->next=NULL;
        if(head==NULL
        )
            head=tail=n;
        else{
            tail-
            >next=n;
            tail=n;}}
    void display(){
        struct
        node*i;int
        sum=0;
        for(i=head;i!=NULL;i=i-
            >next)sum+=i->data;
        printf("%d ",sum);}
END
```

SOURCECODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
int             data;
structnode* next;};
structnode*head=NULL,*tail=NULL;
void insert_end(int val){
structnode*n=(structnode*)malloc(sizeof(structnode));
n->data=val;
n->next=NULL;
if(head==NULL)
head=tail=n;
else{
tail->next=n;
tail=n;}}
void display(){
```

```
        structnode*i;
        int sum=0;
        for(i=head;i!=NULL;i=i->next)
        sum+=i->data;
        printf("%d",sum);}
        int main(){
        int n,val,i;
        printf("Entervalue:");
        scanf("%d",&n);
        for(i=0;i<n;i++){
        scanf("%d",&val);
        insert_end(val);}
        display();
        return0;
    }
```

OUTPUT:

```
Enter value :4
5 10 20 1
36
--------------------------------
Process exited after 10.5 seconds with return value 0
Press any key to continue . . .
```

RESULT:

Thus the program of printing the sum of singly linked list is executed and the output is verified successfully.

Exp. No :2.4(a) Date :	**IMPLEMENTATIONS OF BINARY TREE**

AIM:

To write a C-program for implementing binary tree operations.

Data Structure used: Tree

PSEUDOCODE:

```
BEGIN
        struct node *insert(struct node *r,int k) if(r==NULL)
        struct node *n=(struct node *)malloc(sizeof(struct node));
        n->data=k;
        n->left=n->right=NULL;
        r=n;
        else if(k>r->data)
        r->right=insert(r->right,k);
        else if(k<r->data)
        r->left=insert(r->left,k);
        return r;
        void inorder(struct node *r)
        if(r!=NULL)
        inorder(r->left);
        printf("%d ",r->data);
        inorder(r->right);
        void preorder(struct node *r)
        if(r!=NULL)
        printf("%d ",r->data);
        preorder(r->left);
        preorder(r->right);
        void postorder(struct node *r)
        if(r!=NULL)
        postorder(r->left);
        postorder(r->right);
        printf("%d ",r->data);
        END
```

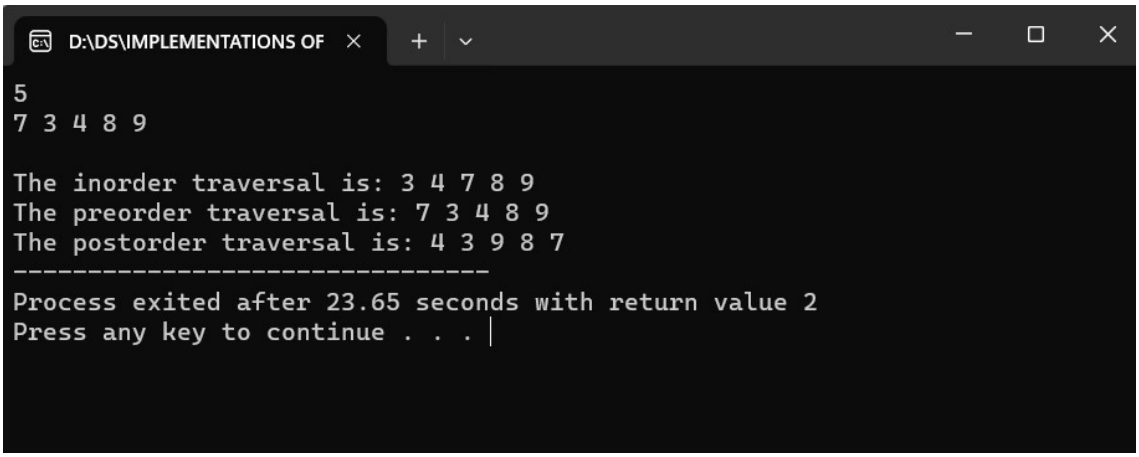
SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *left,*right;
};
struct node *insert(struct node *r,int k)
{
```

```
if(r==NULL){
struct node *n=(struct node *)malloc(sizeof(struct node));
n->data=k;
n->left=n->right=NULL; r=n;
}
else if(k>r->data)
r->right=insert(r->right,k);
else if(k<r->data)
r->left=insert(r->left,k);
return r;
}
void inorder(struct node *r)
{
if(r!=NULL)
{
inorder(r->left);
printf("%d ",r->data);
inorder(r->right);
}
}
void preorder(struct node *r)
{
if(r!=NULL)
{
printf("%d ",r->data);
preorder(r->left);
preorder(r->right);
}
}
void postorder(struct node *r)
{
if(r!=NULL)
{
postorder(r->left);
postorder(r->right);
printf("%d ",r->data);
}
}
int main()
{
struct node *root=NULL;
int n,val,i; scanf("%d",&n); for(i=0;i<n;i++)
{
scanf("%d",&val); root=insert(root,val);
}
printf("\nThe inorder traversal is: "); inorder(root);
printf("\nThe preorder traversal is: "); preorder(root);
printf("\nThe postorder traversal is: "); postorder(root);
}
```

717822F239

OUTPUT:



RESULT:

Thus the program is executed successfully and the output is verified.

Exp. No :2.4(b) Date :	**IMPLEMENTATIONS OF BINARY SEARCH TREE**

AIM:

To write a C-program for implementing binary search tree operations.

Data Structure used: Tree

PSEUDOCODE:

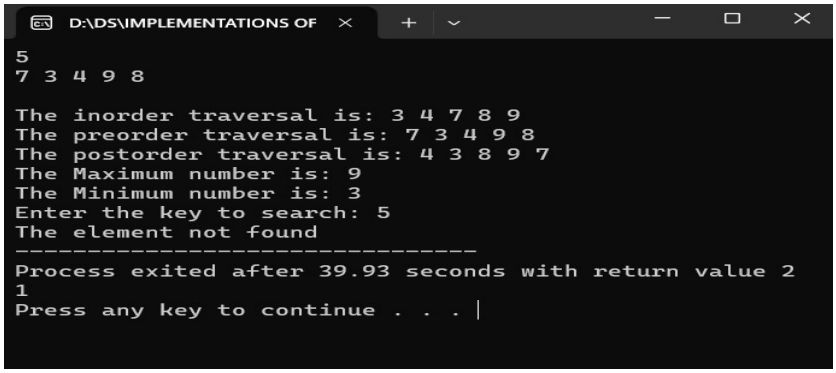
```
BEGIN
  struct node *insert(struct node *r,int k)
    if(r==NULL struct node *n=(struct node *)malloc(sizeof(struct node));
      n->data=k;
      n->left=n->right=NULL;
       r=n;
      else if(k>r->data)
         r->right=insert(r->right,k);
      else if(k<r->data)
          r->left=insert(r->left,k);
           return r;
   void inorder(struct node*r)
      if(r!=NULL)
       inorder(r->left);
          printf("%d ",r->data);
       inorder(r->right);
   void preorder(struct node*r)
     if(r!=NULL)
    printf("%d ",r->data);
      preorder(r->left);
      preorder(r->right);
   void postorder(struct node*r)
    if(r!=NULL)
       postorder(r->left);
       postorder(r->right);
       printf("%d ",r->data);
   int max(struct node *r)
  while(r->right!=NULL)
    r=r->right;
     return r->data;
  int min(struct node *r)
  while(r->left!=NULL)
      r=r->left;
      return r->data;
  struct node *search(struct node *r,int k)
    if(r==NULL)
     return NULL;
  else if(k>r->data)
     return search(r->right,k);
  else if(k<r->data)
     return search(r->left,k);
  else
     return r;
END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
  int data;
struct node *left,*right;
}struct node *insert(struct node *r,int k){
if(r==NULL){
struct node *n=(struct node *)malloc(sizeof(struct node));
n->data=k;
n->left=n->right=NULL;
r=n;
}
else if(k>r->data)
r->right=insert(r->right,k);
   else if(k<r->data)
r->left=insert(r->left,k);
   return r;
}
void inorder(struct node *r){
   if(r!=NULL){
     inorder(r->left);
     printf("%d ",r->data);
     inorder(r->right);
    }
}
void preorder(struct node *r){
    if(r!=NULL){
       printf("%d ",r->data);
       preorder(r->left);
       preorder(r->right);
    }
}
void postorder(struct node *r){
      if(r!=NULL){
        postorder(r->left);
        postorder(r->right);
        printf("%d ",r->data);
    }
}
int max(struct node *r){
  while(r->right!=NULL)
     r=r->right;
     return r->data;
}
int min(struct node *r){
    while(r->left!=NULL)
    r=r->left;
    return r->data;
```

```
 struct node *search(struct node *r,int k){if(r==NULL)
return NULL;
else if(k>r->data)
return search(r->right,k);
else if(k<r->data)
return search(r->left,k);
 else     return r;
}
int main(){
struct node *root=NULL;
int n,val,i;
scanf("%d",&n);
for(i=0;i<n;i++){
scanf("%d",&val);
root=insert(root,val);
}
printf("\nThe inorder traversal is: "); inorder(root);
printf("\nThe preorder traversal is: "); preorder(root);
printf("\nThe postorder traversal is: "); postorder(root);
printf("\nThe Maximum number is: "); printf("%d",max(root));
printf("\nThe Minimum number is: "); printf("%d",min(root));
printf("\nEnter the key to search: ");
scanf("%d",&val);
if(search(root,val)==NULL)
printf("The element not found");
else
printf("The element found");
}
```

OUTPUT:



RESULT:

Thus, the program is executed successfully and the output is verified.

Exp. No :2.5(a) Date :	**SUM OF ALL LEAF NODES**

AIM:

To write a C-program for finding the sum of all leaf nodes.

Data Structure used: Tree

PSEUDOCODE:

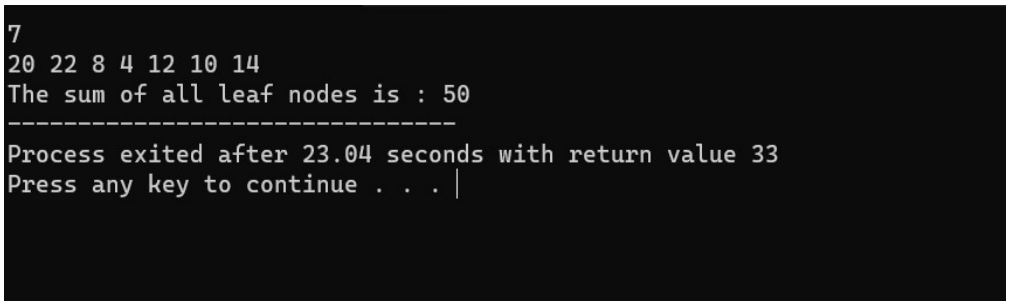
```
BEGIN
        int sum(struct node*r)int s=0;
        if(r==NULL)
          return 0;
        else if(r->left==NULL &&r->right==NULL)
           return r->data;
        else
           return sum(r->left)+sum(r->right);
END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *left,*right;
};
struct node *insert(struct node *r,int key)
{
if(r==NULL)
{
struct node *n=(struct node *)malloc(sizeof(struct node));
n->data=key;
n->left=n->right=NULL;
r=n;
}else if(key>r->data)
   r->right=insert(r->right,key);
 else if(key<r->data)
   r->left=insert(r->left,key);
   return r;
}
int sum(struct node *r)
{
int s=0;
if(r==NULL)
  return 0;
else if(r->left==NULL &&r->right==NULL)
   return r->data;
else
  return sum(r->left)+sum(r->right);
}
int main()
```

```
{
struct node *root=NULL;
  int n,val,i;
  scanf("%d",&n); for(i=0;i<n;i++)
{
scanf("%d",&val);
root=insert(root,val);
}
printf("The sum of all leaf nodes is : %d",sum(root));
}
```

OUTPUT:



RESULT:

Thus the program is executed successfully and the output is verified.

Exp. No :2.5(b) Date :	**NUMBERS LIE IN A RANGE**

AIM:

To write a C-program for finding the count of numbers that lie in a certain a range.
Data Structure used: Tree

PSEUDOCODE:

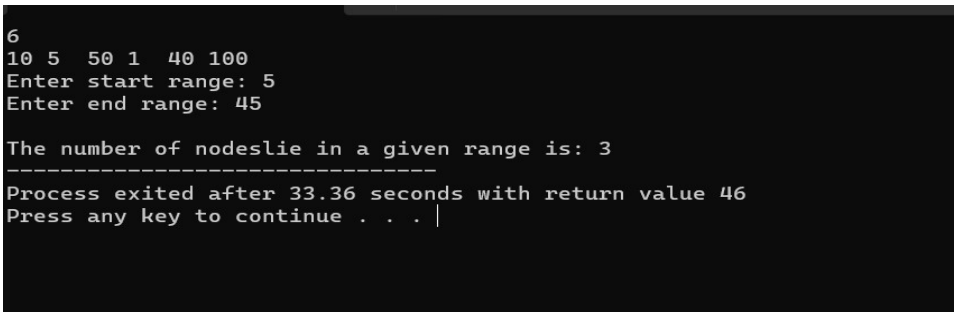
```
BEGIN
count(struct node *r,int s,int e)static int c=0;
  if(r!=NULL)
    count(r->left,s,e);
  if(r->data>=s && r->data<=e)
      c++;
      count(r->right,s,e);
     return c;
END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *left,*right;
};
struct node *insert(struct node *r,int k)
{
if(r==NULL)
{
struct node *n=(struct node *)malloc(sizeof(struct node));
  n->data=k;
   n->left=n->right=NULL; r=n;
}
else if(k>r->data)
  r->right=insert(r->right,k);
else if(k<r->data)
   r->left=insert(r->left,k);
return r;
}
int count(struct node *r,int s,int e)
{
static int c=0;
if(r!=NULL)
{
  count(r->left,s,e);
if(r->data>=s && r->data<=e) c++;
  count(r->right,s,e);
}
return c;
}
```

```
int main()
{
struct node *root=NULL;
int n,val,i,s,e;
   scanf("%d",&n);
for(i=0;i<n;i++)
{
   scanf("%d",&val);
     root=insert(root,val);
}
   printf("Enter start range: ");
 scanf("%d",&s);
 printf("Enter end range: ");
 scanf("%d",&e);
printf("\nThe number of nodeslie in a given range is: %d",count(root,s,e));
 }
```

OUTPUT:



RESULT:

Thus, the C program to search the given element in the given array using binary search is successfully executed and the output is verified .