```
/*LINEAR SEARCH AND BINARY SEARCH*/
#include <stdio.h>
#include <stdlib.h>
int A[100], n, key;
int lsearch (int A[], int n, int key)
{
int i = -1;
while (i < n) /* Traversing till the end of table */
{
if (A[++i]==key)
/* element is found */
return i;
}
return -1;
/* item is not found */
}
```

```
int binsearch (int A[], int n, int key)
{
int first, last, mid, i;
first = 0;
last = n -1;
while (first<=last)
{
mid = (first + last)/2;
 if (key==A[mid]) return mid;
```

```
else
if (key < A[mid])
last= mid- 1;
else
first = mid + 1;
}
return -1;
}
void acceptInput()
{
int i;
printf("Enter Number of Elements :");
scanf ("%d", &n);
for (i= 0; i<n; i++)
{
 printf ("Enter Element %d: ", i);
scanf ("%d",&A[i]);
}
printf ("Enter an Element to be Searched: ");
scanf ("%d",&key);
}
void main()
{
int ch, flag;
while (1)
{
printf("\n Searching Techniques");
printf("\n**********");
printf("\n 1. Linear Search ");
```

```
printf("\n 2. Binary Search ");
printf("\n 3. Exit ");
printf("\n Enter your choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1: acceptInput();
flag=lsearch(A, n, key);
if(flag==-1)
printf("\n Search is Unsuccessful.");
else
printf("\n An Element %d Found at Position: %d", key, flag);
break;
 case 2: printf("\n Enter Elements in Ascending Order for Binary Search\n");
acceptInput();
flag = binsearch(A,n,key);
if ( flag ==-1)
printf("%d not found in array \n",key );
else
printf("An Element %d Found at Position :%d  \n ",key , flag);
break;
case 3: exit(0);
}
}
}
```

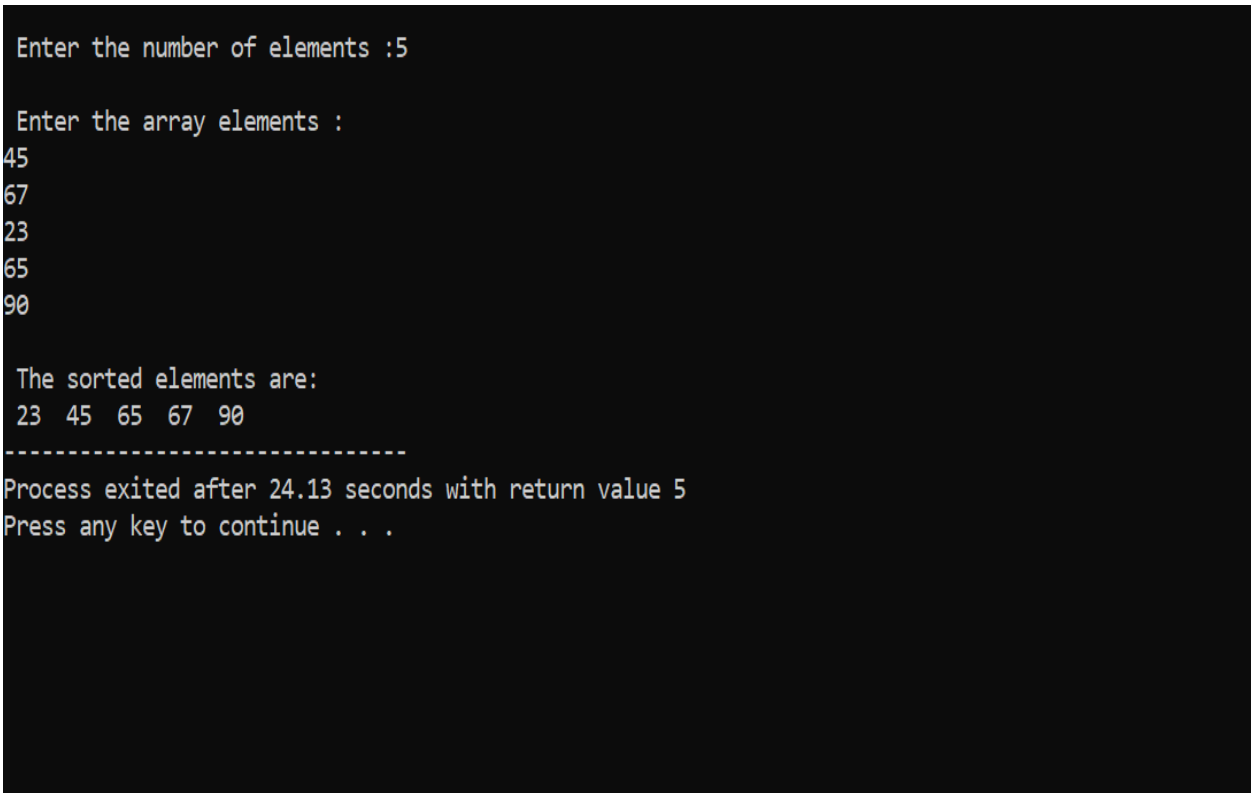
OUTPUT:

```
 Searching Techniques
**********
 1. Linear Search
 2. Binary Search
 3. Exit
 Enter your choice: 1
Enter Number of Elements :4
Enter Element 0: 34
Enter Element 1: 56
Enter Element 2: 78
Enter Element 3: 9
Enter an Element to be Searched: 9

 An Element 9 Found at Position: 3
 Searching Techniques
**********
 1. Linear Search
 2. Binary Search
 3. Exit
 Enter your choice: 2

 Enter Elements in Ascending Order for Binary Search
Enter Number of Elements :3
Enter Element 0: 67
Enter Element 1: 23
Enter Element 2: 45
Enter an Element to be Searched: 45
An Element 45 Found at Position :2

 Searching Techniques
**********
 1. Linear Search
 2. Binary Search
 3. Exit
 Enter your choice:
```

```
/*BUBBLE SORT*/
#include<stdio.h>
void bubble_sort(int a[], int n)
{
int pass, temp, j ;
for (pass = 1; pass < n; pass ++)
{
for (j = 0; j <= n- pass - 1; j++)
{
if (a [j] > a[j + 1])
{
temp = a [j];
a[j] = a [j+ 1];
a[j + 1]= temp;
}
}
}
}
int main()
{
int i,j,a [20], n, temp;
printf("\n Enter the number of elements :");
scanf("%d",&n);
printf ("\n Enter the array elements : \n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
bubble_sort (a,n);
printf ("\n The sorted elements are: \n");
for(i=0;i<n; i++)
```

```
printf (" %d ",a[i]);
}
```

OUTPUT



```
/* INSERTION SORT AND SELECTION SORT*/
#include<stdio.h>
#include<stdlib.h>
int a[100],n;
/* to find the location of max element */
int max(int a[], int k, int n)
{
int loc, j, max;
max = a[k];
loc = k;
for (j=k+ 1; j <= n - 1; j++)
if (max<a[j])            /* use max>a[j] for ascending order */
{
max = a[j];
loc = j;
}
return (loc);
}
void insertion_sort (int a[], int n)
{
int pass, k, temp, j;
for (pass=1; pass<n; pass++)
{
k = a[pass]; /* k is to be inserted at proper place */
for (j=pass-1; j>=0 && k>a[j]; j--)
a[j+1] = a[j];
a[j+1]=k;
}
}
```

```
void acceptinput()
{
int i;
printf("enter the number of elements:\n ");
scanf("%d",&n);
printf("\n enter the array elements :\n ");
for(i=0; i<n; i++)
scanf("%d",&a[i]);
}
void display()
{
int i;
printf("\n the sorted array is: ");
for(i=0;i<n;i++)
printf(" %d ", a[i]);
}
void main ()
{
int k, temp, loc, ch;
while (1)
{
printf("\n sorting techniques");
printf("\n**********");
printf("\n 1. insertion sort ");
printf("\n 2. selection sort ");
printf("\n 3. exit ");
printf("\n enter your choice :\n ");
scanf("%d",&ch);
switch(ch)
```

```
{
case 1:
acceptinput();
insertion_sort(a,n);
display();
break;
case 2:
acceptinput();
for(k=0; k<n; k++)
{
loc=max(a, k,n);
temp=a[k];
a[k]=a[loc];
a[loc]=temp;
}
display();
break;
case 3: exit(0);
}
}
}
```

OUTPUT

```
 sorting techniques
**********
 1. insertion sort
 2. selection sort
 3. exit
 enter your choice :
 1
enter the number of elements:
 4

 enter the array elements :
 56
34
10
25

 the sorted array is:  56  34  25  10
 sorting techniques
**********
 1. insertion sort
 2. selection sort
 3. exit
 enter your choice :
 2
enter the number of elements:
 3

 enter the array elements :
 45
1
89

 the sorted array is:  89  45  1
```

```
 sorting techniques
**********
 1. insertion sort
 2. selection sort
 3. exit
 enter your choice :
 3

--------------------------------
Process exited after 63.89 seconds with return value 0
Press any key to continue . . .
```

/* 4. Single Linked List*/

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<ctype.h>
struct node
{
int INFO;
struct node *LINK;
};
typedef struct node NODE;
NODE *start=NULL;
void create()
{
char ch;
int i=0;
NODE *CURPTR,*NEWNODE;
CURPTR=(NODE *)malloc(sizeof(NODE));
start=CURPTR;
while(1)
{
printf("\n Enter the node %d:",i+1);
scanf("%d",&CURPTR->INFO);
printf("\n Do you wish to add one more node(Y/N):");
ch=getche();
if (toupper(ch)=='Y')
{
NEWNODE=(NODE*)malloc(sizeof(NODE));
CURPTR->LINK=NEWNODE;
CURPTR=NEWNODE;
}
```

```
else
{
CURPTR->LINK=NULL;
break;
}
i++;
}
}

void display()
{
NODE *CURPTR =start;
if(start==NULL)
printf("\n The linked list is empty");
else
{
while(CURPTR != NULL)
{
printf("%d",CURPTR->INFO);
printf("->");
CURPTR=CURPTR->LINK;
}
}
}

void insert_node(int item)
{
NODE *NEWNODE;
NEWNODE=(NODE*)malloc(sizeof(NODE));
NEWNODE->INFO=item;
NEWNODE->LINK=start;
```

```
start=NEWNODE;
}
void delete_node()
{
NODE *CURPTR;
if(start==NULL)
{
printf("\n The linked list is empty \n");
return;
}
else
{
CURPTR=start;
start=start->LINK;
free(CURPTR);
}
}
int main()
{
int ch,item,pos;
while(1)
{
printf("\n 1.Create a linked list");
printf("\n 2.Display the list");
printf("\n 3.Insert node");
printf("\n 4.Delete node");
printf("\n 5.Exit");
printf("\n Enter your choice:");
scanf("%d",&ch);
switch(ch)
{
```

```
case 1:start=NULL;
create();
break;
case 2:display();
break;
case 3:printf("\n Enter the item to insert:");
scanf("%d",&item);
printf("\n Linked list before insertion is:\n");
display();
insert_node(item);
printf("\n Linked list after Insertion is:\n");
display();
break;
case 4:printf("\n Linked list before deletion is :\n");
display();
delete_node();
printf("\n Linked list after deletion is :\n");
display();
break;
case 5:exit(0);
}
}
}
```

OUTPUT

```
 1.Create a linked list
 2.Display the list
 3.Insert node
 4.Delete node
 5.Exit
 Enter your choice:
1

 Enter the node 1:47

 Do you wish to add one more node(Y/N):Y
 Enter the node 2:56

 Do you wish to add one more node(Y/N):N
 1.Create a linked list
 2.Display the list
 3.Insert node
 4.Delete node
 5.Exit
 Enter your choice:2
47->56->
 1.Create a linked list
 2.Display the list
 3.Insert node
 4.Delete node
 5.Exit
```

```
 Enter your choice:
3

 Enter the item to insert:90

 Linked list before insertion is:
47->56->
 Linked list after Insertion is:
90->47->56->
 1.Create a linked list
 2.Display the list
 3.Insert node
 4.Delete node
 5.Exit
 Enter your choice:4

 Linked list before deletion is :
90->47->56->
 Linked list after deletion is :
47->56->
 1.Create a linked list
 2.Display the list
 3.Insert node
 4.Delete node
 5.Exit
```

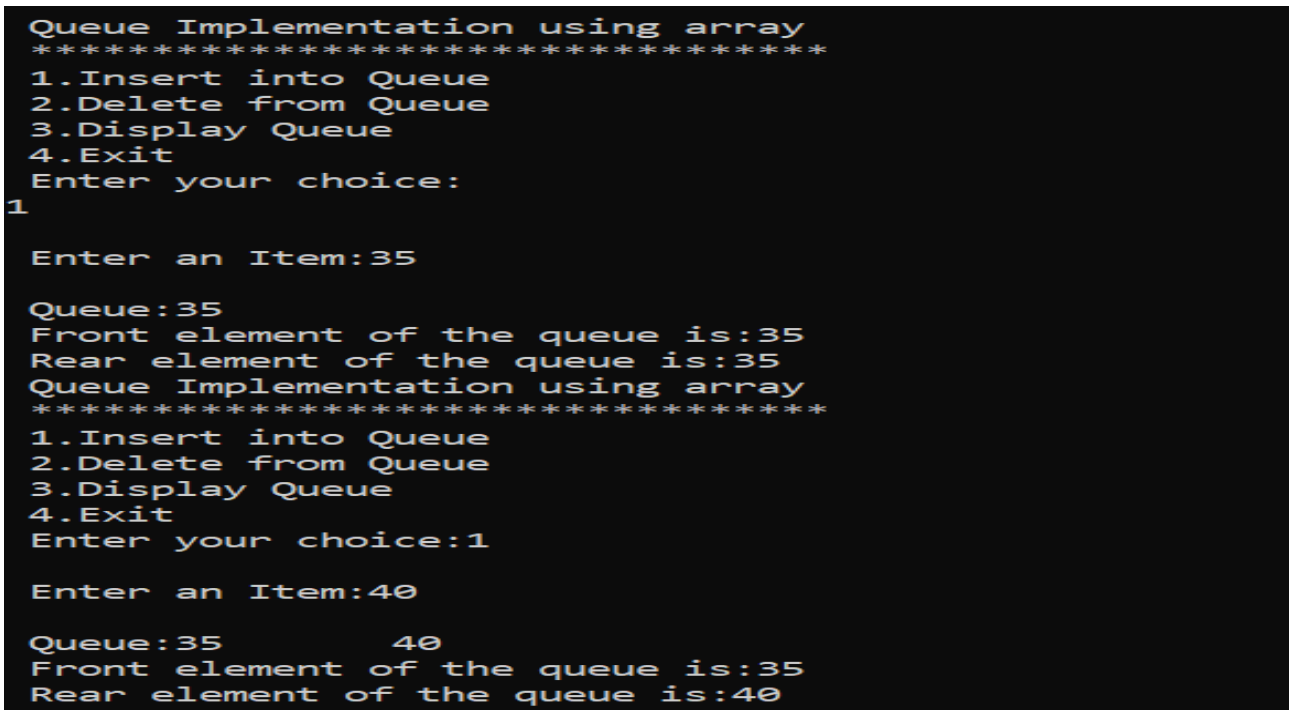
/*LINEAR QUEUE*/

```
#include<stdio.h>
#include<stdlib.h>
#define N 10
int QUEUE[N],FRONT=0,REAR=-1,ITEM;
void Qinsert()
{
if(REAR==N-1)
printf("\n Queue Overflow");
else
{
printf("\n Enter an Item:");
scanf("%d",&ITEM);
REAR++;
QUEUE[REAR]=ITEM;
}
}
void Qdelete()
{
if(REAR==FRONT-1)
printf("\n Queue Underflow");
else if(REAR==FRONT)
{
printf("\n This is the Last Element in the Queue");
printf("\n The Last Element Deleted is:%d",QUEUE[FRONT]);
FRONT=0;
REAR=-1;
}
else
{
printf("\n Deleted item is %d",QUEUE[FRONT]);
```

```
FRONT++;
}
}
void Qdisplay()
{
int i;
if(REAR==FRONT-1)
printf("\n\t No elements in Queue");
else
{
printf("\n Queue:");
for(i=FRONT;i<=REAR;i++)
{
printf("%d\t",QUEUE[i]);
}
printf("\n Front element of the queue is:%d",QUEUE[FRONT]);
printf("\n Rear element of the queue is:%d",QUEUE[REAR]);
}
}
void main()
{
int ch;
while(1)
{
printf("\n Queue Implementation using array");
printf("\n ********************************");
printf("\n 1.Insert into Queue");
printf("\n 2.Delete from Queue");
printf("\n 3.Display Queue");
printf("\n 4.Exit");
printf("\n Enter your choice:");
```

```
scanf("%d",&ch);
switch(ch)
{
case 1:Qinsert();
Qdisplay();
break;
case 2:Qdelete();
Qdisplay() ;
break;
case 3:Qdisplay();
break;
case 4:exit(0);
}
}
}
```

OUTPUT



```
Queue Implementation using array
*********************************
1.Insert into Queue
2.Delete from Queue
3.Display Queue
4.Exit
Enter your choice:1

Enter an Item:70

Queue:35        40      70
Front element of the queue is:35
Rear element of the queue is:70
Queue Implementation using array
*********************************
1.Insert into Queue
2.Delete from Queue
3.Display Queue
4.Exit
Enter your choice:2

Deleted item is 35
Queue:40        70
Front element of the queue is:40
Rear element of the queue is:70
Queue Implementation using array
*********************************
1.Insert into Queue
2.Delete from Queue
3.Display Queue
4.Exit
Enter your choice:3

Queue:40        70
Front element of the queue is:40
Rear element of the queue is:70
```

```
Queue Implementation using array
*********************************
1.Insert into Queue
2.Delete from Queue
3.Display Queue
4.Exit
Enter your choice:4

--------------------------------
Process exited after 71.61 seconds with return value 0
Press any key to continue . . .
```

```
/*CIRCULAR QUEUE*/
#include<stdio.h>
 #include<stdlib.h>
#define N 10
int QUEUE [N], FRONT=-1, REAR=-1, ITEM;
/*queue is initially empty*/
/* Function to insert an ITEM into a circular queue */
void CQinsert()
{

if ((FRONT==(REAR+1) % N))
/* to check overflow condition */
printf("\n Queue Overflow\n");
else
{
printf("\n Enter the Element to be Inserted:\n");
 scanf("\n %d", &ITEM);
if (FRONT==-1)
{

        FRONT=0;
        REAR=0;
}
else
{
        REAR=(REAR+1)%N;
}
QUEUE [REAR] =ITEM;
}
}
/* Function to display all the elements in a circular queue */
```

```
 void CQdisplay()
{
int i;
if (FRONT==-1)
printf("no elements in queue\n");
else
{
printf("\n Circular Queue : \n");
if (FRONT<=REAR)
{
for (i=FRONT; i<=REAR;i++)
printf("\t%d", QUEUE[i]);
}

if (FRONT>REAR)
{
for (i=FRONT;i<=N-1;i++)
printf("\t%d", QUEUE [i]);
for (i=0;i<=REAR; i++)
printf("\t%d", QUEUE[i]);
}
}

printf("\n Front Element of the CQueue is: %d", QUEUE [FRONT]);
 printf("\n Rear Element of the CQueue is : %d", QUEUE [REAR]);
}
/* Function to delete an ITEM from a circular queue*/
void CQdelete()
{
if (FRONT==-1)
printf("\n Queue Underflow");
```

```
else
{

ITEM=QUEUE [FRONT];
printf("\n The Deleted Item is: %d", QUEUE [FRONT]);
if(FRONT==REAR)
{

FRONT=-1;
REAR=-1;
}
else
FRONT=(FRONT+1)%N;
/* if there is only one element in a queue */ /* after deleting FRONT and REAR should set to -1 */
/* increment FRONT value */
}
}
void main()
{
int ch;
while(1)
{
printf("\n Circular Queue implementation using Array");
printf("\n *****");
printf("\n 1. Circular Queue Insert ");
printf("\n 2. Circular Queue Delete ");
printf("\n 3. Circular Queue Display ");
printf("\n 4. Exit ");
printf("\n Enter your Choice: \n");
scanf("%d", &ch);
switch(ch)
```

```
{
case 1:
CQinsert();
//CQdisplay();
break;
case 2:
CQdelete();
//CQdisplay();
break;
case 3:
CQdisplay();
break;
case 4:
exit(0);
} } }
```

OUTPUT

```
 Circular Queue implementation using Array
 *****
 1. Circular Queue Insert
 2. Circular Queue Delete
 3. Circular Queue Display
 4. Exit
 Enter your Choice:
1

 Enter the Element to be Inserted:
50

 Circular Queue implementation using Array
 *****
 1. Circular Queue Insert
 2. Circular Queue Delete
 3. Circular Queue Display
 4. Exit
 Enter your Choice:
1

 Enter the Element to be Inserted:
60

 Circular Queue implementation using Array
 *****
 1. Circular Queue Insert
 2. Circular Queue Delete
 3. Circular Queue Display
 4. Exit
 Enter your Choice:
3

 Circular Queue :
        50      60
 Front Element of the CQueue is: 50
 Rear Element of the CQueue is : 60
 Circular Queue implementation using Array
 *****
```

```
 Circular Queue implementation using Array
 *****
 1. Circular Queue Insert
 2. Circular Queue Delete
 3. Circular Queue Display
 4. Exit
 Enter your Choice:
2

 The Deleted Item is: 50
 Circular Queue implementation using Array
 *****
 1. Circular Queue Insert
 2. Circular Queue Delete
 3. Circular Queue Display
 4. Exit
 Enter your Choice:
3

 Circular Queue :
        60
 Front Element of the CQueue is: 60
 Rear Element of the CQueue is : 60
 Circular Queue implementation using Array
 *****
 1. Circular Queue Insert
 2. Circular Queue Delete
 3. Circular Queue Display
 4. Exit
 Enter your Choice:
4

--------------------------------
Process exited after 37.35 seconds with return value 0
Press any key to continue . . .
```

/*7. ORDERED SINGLE LINKED LIST*/

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int INFO;
struct node *LINK;
};

typedef struct node NODE;
NODE *start=NULL;
void insertOrdered (int data)
{
NODE *NEWNODE= (NODE *)malloc(sizeof(NODE));
NEWNODE->INFO = data;
if(start==NULL)
{
start = NEWNODE;
start->LINK=NULL;
}
else if(data < start->INFO)
{
NEWNODE->LINK=start;
start=NEWNODE;
}
else
{
NODE *PREVPTR=start;
NODE *CURRPTR = start->LINK;
while (CURRPTR != NULL && data > CURRPTR->INFO)
{
```

```
PREVPTR=CURRPTR;
CURRPTR=CURRPTR->LINK;
}
PREVPTR->LINK = NEWNODE;
NEWNODE->LINK = CURRPTR;
}
}
void deleteOrdered (int data)
{
NODE *PREVPTR = start;
NODE *CURRPTR = start->LINK;
if(start==NULL)
printf("\n List is Empty");
else
if(data == start->INFO)
{
start=CURRPTR;
free (PREVPTR);
}
else
{
while (CURRPTR != NULL && CURRPTR->INFO != data)
{
PREVPTR = CURRPTR;
CURRPTR = CURRPTR->LINK;
}
if (CURRPTR != NULL)
{
PREVPTR->LINK = CURRPTR->LINK;
free (CURRPTR);
}
```

```
else
printf("\n Data Not Found in the List");
}
}

void display()
{
NODE *CURRPTR = start;
if (CURRPTR == NULL)
printf(" Empty ");
else
{
while (CURRPTR != NULL)
{
printf("%10d", CURRPTR -> INFO);
CURRPTR = CURRPTR -> LINK;
}
}
}
int main()
{
int ch,data;
while(1)
{
printf("\n Ordered Linked List Operations");
printf("\n *******");
printf("\n 1. Insert");
printf("\n 2. Delete");
printf("\n 3. Display");
printf("\n 4. Exit");
printf("\n Enter Your Choice: ");
```

```
scanf("%d", &ch);
switch(ch)
{
case 1: printf("\n Enter Data to be Inserted : ");
scanf("%d", &data);
printf("\n Linked List before Insertion is: \n");
display();
insertOrdered (data);
printf("\n Linked List after Insertion is: \n");
display();
break;
case 2: printf("\n Enter Data to be Deleted : ");
scanf("%d", &data);
printf("\n Linked List before Deletion is: \n");
display();
deleteOrdered (data);
printf("\n Linked List after Deletion is: \n");
display();
break;
case 3: display();
break;
case 4:
exit(0);
}
}
}
```

OUTPUT

```
Ordered Linked List Operations
*******
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1

Enter Data to be Inserted : 200

Linked List before Insertion is:
Empty
Linked List after Insertion is:
      200
Ordered Linked List Operations
*******
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1

Enter Data to be Inserted : 100

Linked List before Insertion is:
      200
Linked List after Insertion is:
      100       200
Ordered Linked List Operations
*******
1. Insert
2. Delete
3. Display
4. Exit
```

```
Enter Your Choice: 1

Enter Data to be Inserted : 300

Linked List before Insertion is:
      100       200
Linked List after Insertion is:
      100       200       300
Ordered Linked List Operations
*******
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 2

Enter Data to be Deleted : 300

Linked List before Deletion is:
      100       200       300
Linked List after Deletion is:
      100       200
Ordered Linked List Operations
*******
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 3
      100       200
Ordered Linked List Operations
*******
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 4

--------------------------------
Process exited after 45.19 seconds with return value 0
```

/*8.polynomial addition*/

```
#include<stdio.h>
#include<stdlib.h>
struct poly{
   int coeff;
   int pow;
   struct poly *next;
};
typedef struct poly NODE;
NODE *poly1=NULL;
NODE *poly2=NULL;
NODE *poly3=NULL;
 NODE *news=NULL;
 NODE *end=NULL;
 int item;
 void addpoly()
{
     int i,p;
     printf("Enter highest power for x :\n");
     scanf("%d",&p);
     printf("\nFirst Polynomial :\n");
     for(i=p;i>=0;i--)
     {
           news= (NODE *)malloc(sizeof(NODE));
     news->pow=p;
     printf("Enter Co-efficient for degree %d\n",i);
     scanf("%d",&item);
     news->coeff=item;
     news->next=NULL;
     if(poly1==NULL)
           poly1=news;
```

```
 else
        end->next=news;
 end=news;
 }
printf("\n\nSecond Polynomial:\n");
 end=NULL;
 for(i=p;i>=0;i--)
 {
news= (NODE *)malloc(sizeof(NODE));
        news->pow=p;
 printf("Enter Co-efficient for degree %d\n",i);
 scanf("%d",&item);
 news->coeff=item;
 news->next=NULL;
 if(poly2==NULL)
        poly2=news;
 else
        end->next=news;
 end=news;
 }
NODE *p1=poly1,*p2=poly2;
 end=NULL;
 while(p1 !=NULL && p2!=NULL)
  {
       if(p1->pow == p2->pow){
                NODE *news= (NODE*)malloc(sizeof(NODE));
          news->pow=p--;
          news->coeff=p1->coeff + p2->coeff;
          news->next=NULL;
          if(poly3==NULL)
            poly3=news;
```

```
            else
              end->next=news;
            end=news;
          }
          p1=p1->next;
          p2=p2->next;
     }
}

void  display(){
  NODE *t=poly3;
  printf("\n\nAnswer after addition is : ");
  while(t!=NULL){
     printf("%d",t->coeff);
      printf("X^%d",t->pow);
     t=t->next;
     if(t!=NULL)
     printf("+");
     else
     printf(" ");
  }
}
int main(){
   addpoly();
   display();
   getch();
}
```

OUTPUT:

```
Enter highest power for x :
3

First Polynomial :
Enter Co-efficient for degree 3
4
Enter Co-efficient for degree 2
3
Enter Co-efficient for degree 1
2
Enter Co-efficient for degree 0
1


Second Polynomial:
Enter Co-efficient for degree 3
4
Enter Co-efficient for degree 2
3
Enter Co-efficient for degree 1
2
Enter Co-efficient for degree 0
1


Answer after addition is : 8X^3+6X^2+4X^1+2X^0
```

```
/*STACK USING ARRAY*/
#include<stdio.h>
#include<stdlib.h>
#define MAXSTK 5
int TOP = -1;
int s[MAXSTK];
void push();
void pop();
void display();
int main()
{
int choice;
while(1)
{
printf("1. Push\n");
printf("2.Pop\n");
printf("3.Display\n");
printf("4.Quit\n");
printf("Enter your choice: ");
 scanf("%d", &choice);
switch(choice)
{
case 1:
 push();
break;
case 2:
pop();
break;
case 3:
 display();
```

```
break;
case 4:
exit(0);
default: printf("Wrong choice\n");
 } /*End of switch*/
}
}
/* Function to push an push()*/
void push()
{

int item;
if (TOP== (MAXSTK-1))
printf("Stack Overflow\n");
else
{
/* 'item' is the item to be pushed */
printf("Enter the item to be pushed in stack : ");
scanf("%d", &item);
TOP=TOP+1;
s[TOP] = item;
}
}
/* Function to pop an item from the stack */
void pop()
{
if (TOP == -1)
printf("Stack Underflow\n");
else
{
```

```
printf("Popped element is: %d\n",s[TOP]);

TOP=TOP-1;

}

}

void display()

{

int i;
if(TOP == -1)
printf("Stack Underflow\n");
else
{

printf("Stack elements : \n");
for(i= TOP; i >=0; i--)
printf("%d\n", s[i] );
}
}
```

OUTPUT:

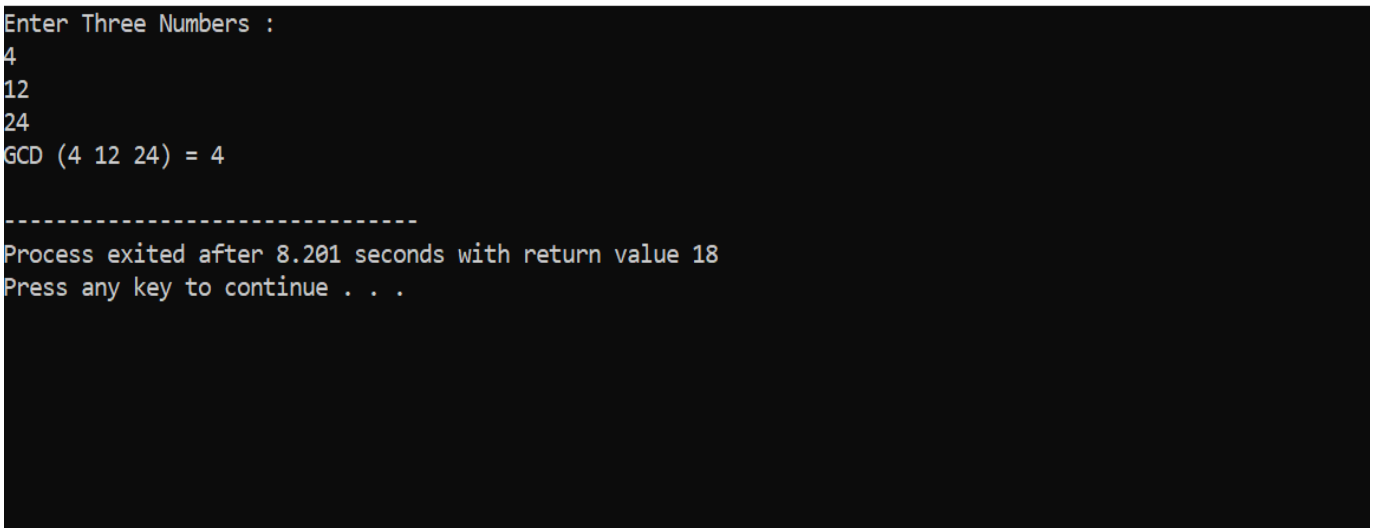
```
1. Push
2.Pop
3.Display
4.Quit
Enter your choice: 1
Enter the item to be pushed in stack : 40
1. Push
2.Pop
3.Display
4.Quit
Enter your choice: 1
Enter the item to be pushed in stack : 50
1. Push
2.Pop
3.Display
4.Quit
Enter your choice: 1
Enter the item to be pushed in stack : 60
1. Push
2.Pop
3.Display
4.Quit
Enter your choice: 3
Stack elements :
60
50
40
1. Push
2.Pop
3.Display
4.Quit
Enter your choice: 2
Popped element is: 60
```

```
1. Push
2.Pop
3.Display
4.Quit
Enter your choice: 3
Stack elements :
50
40
1. Push
2.Pop
3.Display
4.Quit
Enter your choice: 4

--------------------------------
Process exited after 31.23 seconds with return value 0
Press any key to continue . . .
```

/*GCD of three numbers*/

```
#include<stdio.h>
int gcd(int m,int n)
{
        if (n==0)
        return(m);
        else
         if (n> m)
         return(gcd (n, m));

else
return (gcd (n, m % n));
}
void main()
{

int k, m, n;
printf ("Enter Three Numbers :\n");
scanf("%d %d %d", &k, &m, &n );
printf ("GCD (%d %d %d) = %d\n", k, m, n, gcd (k, gcd (m, n)));
}
```

OUTPUT



/*CIRCULAR QUEUE USING LINKED LIST*/

```
#include<stdio.h>
#include<stdlib.h>
struct queue
{
int info;
struct queue *link;
};
struct queue *front= NULL, *rear = NULL;
void QInsert(int item)
{
struct queue *new_node;
new_node = (struct queue*)malloc(sizeof(struct queue));
new_node-> info = item;
new_node-> link = NULL;
if(front==NULL && rear == NULL)
{
front = rear = new_node;
rear -> link = front;
}
else
{
rear -> link = new_node;
rear =new_node;
rear -> link = front;
}
}
void QDelete()
{
struct queue *ptr;
ptr=front;
```

```
if(front == NULL && rear == NULL)
printf("\n Queue is Empty");
else
 if(front == rear)
{
        front=rear=NULL;
        printf("\n The value being deleted is : %d", ptr -> info);
         free (ptr);
}
else
{
        front=front->link;
        rear->link=front;
printf("\n The value being deleted is: %d", ptr -> info);
free (ptr);
}
}
```

```
void Display()
{

struct queue *ptr; ptr = front;
if(front == NULL && rear == NULL)
printf("\n Queue is Empty");
else
{
printf("\n The Queue Elements are: ");
do
```

```
{
printf("%d ", ptr -> info);
ptr = ptr -> link;
}while(ptr != front);
}
}
void main()
{
int val, choice;
do
{
printf("\n *****MAIN MENU*****");
printf("\n 1. Insert");
printf("\n 2. Delete");
printf("\n 3. Display");
 printf("\n 4. Exit");
printf("\n Enter your option: ");
scanf("%d", &choice);
switch(choice)
{
case 1:
printf("\n Enter the number to insert into Queue : "); scanf("%d", &val);
QInsert(val);
break;
case 2:
QDelete();
break;
case 3:
Display();
 break;
}
}while (choice!=4);

}
```

OUTPUT

```
*****MAIN MENU*****
1. Insert
2. Delete
3. Display
4. Exit
Enter your option:
1

Enter the number to insert into Queue : 30

*****MAIN MENU*****
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 1

Enter the number to insert into Queue : 50

*****MAIN MENU*****
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 1

Enter the number to insert into Queue : 60

*****MAIN MENU*****
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 3

The Queue Elements are: 30 50 60
```

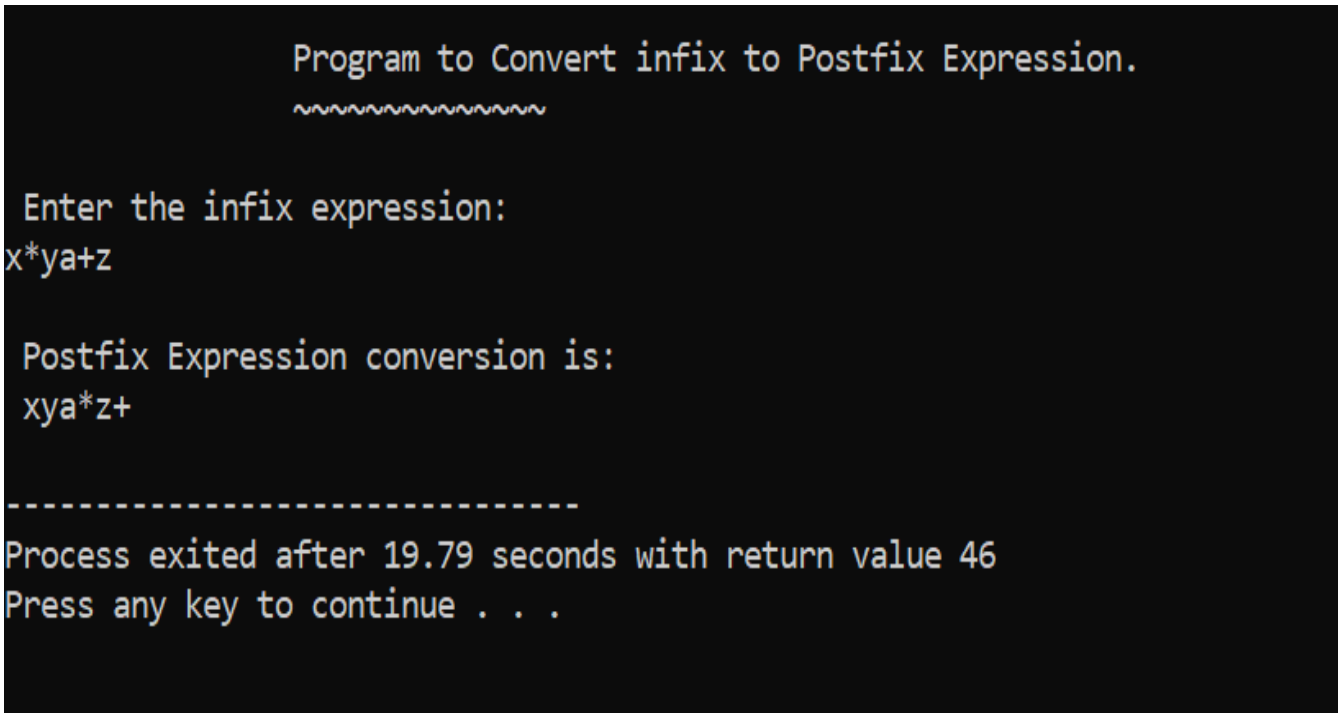
```
*****MAIN MENU*****
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 2

The value being deleted is: 30
*****MAIN MENU*****
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 3

The Queue Elements are: 50 60
*****MAIN MENU*****
1. Insert
2. Delete
3. Display
4. Exit
Enter your option: 4

--------------------------------
Process exited after 43.27 seconds with return value 4
Press any key to continue . . .
```

```
/*INFIX TO POSTFIX CONVERSION*/
#include<stdio.h>
#include<string.h>
#define MAX 20
char s[MAX];
int top=0;
int precedence (char elem);
void push(char elem);
int pop();
int main()
{
char infix [MAX], postfix [MAX], ch,elem;
int i=0,j=0;
printf("\n\t\tProgram to Convert infix to Postfix Expression.");
printf("\n\t\t~~~~~~~~~~~~~~\n");
printf("\n Enter the infix expression: \n");
scanf("%s", infix);
push('#');
for(i=0;i<strlen(infix); i++)
{
ch=infix[i];
 if(isalnum(ch))
postfix[j++]=ch;
else
if(ch=='(')
 push(ch);
 else
 if(ch==')')
{
while(s[top]!='(')
postfix[j++] = pop();
```

```
elem = pop();
}
else
{
while (precedence (s[top])>= precedence(ch))
postfix[j++]=pop();
push(ch);
}
}/*End of scanning elements */
while(s[top] != '#')
postfix[j++]=pop();
postfix[j]='\0';
printf("\n Postfix Expression conversion is: \n %s \n", postfix);
}
/* Function for Precedence */
int precedence (char elem)
{
switch(elem)
{

case '+':
case '-':
        return(1);
case '*':
case '/':
return(2);

case '^': return(3);
case '(':
case '#': return(0);
}
```

```
}

/* Function to void push (char elem)*/
void push(char elem)
{
++top;
s[top]=elem;
}
/* Function for poping Character */
int pop()
{
char elem;
elem=s[top];
--top;
return(elem);
}
```

OUTPUT

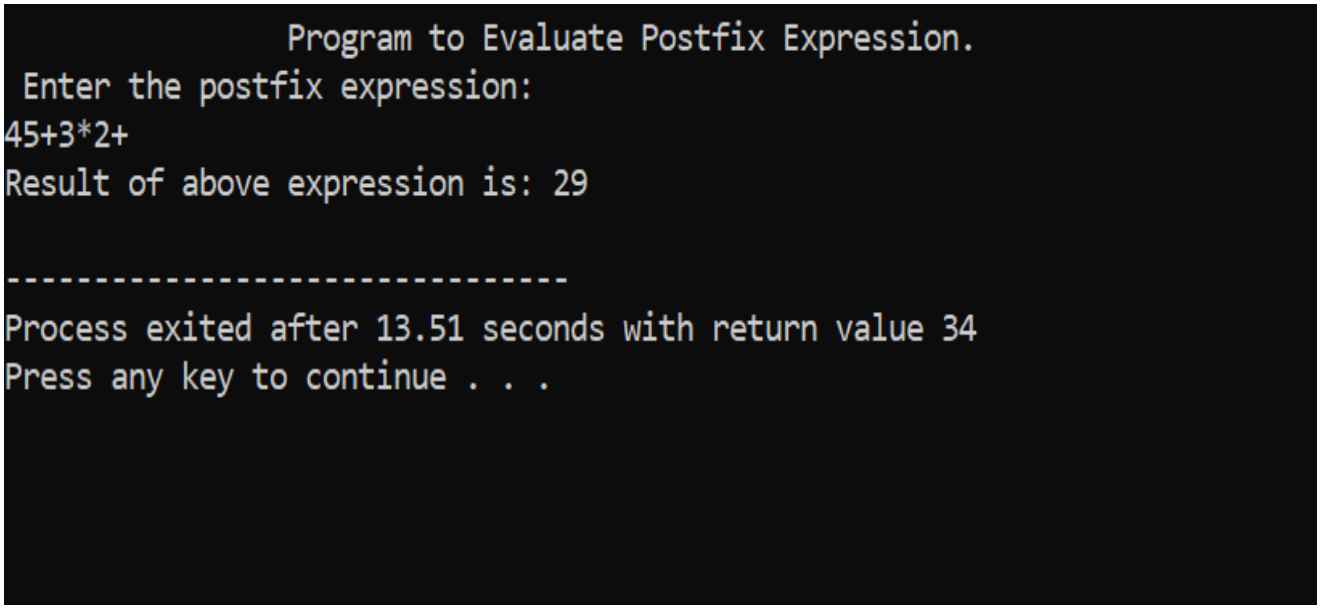


/*postfix expression evaluation*/

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<ctype.h>
#define MAX 20
int s[MAX],top=0;
void push(int ch);
int pop();
void main()
{
char postfix[MAX], ch;
int i,op1, op2, res;
printf("\n\t\tProgram to Evaluate Postfix Expression.");
printf("\n Enter the postfix expression: \n");
scanf("%s",postfix);
for (i=0;i<strlen(postfix); i++)
{
ch=postfix[i];
if(isdigit(ch))/* Check whether digit */
push(ch-'0');
else
{
op2=pop();
op1=pop();
switch(ch)
{
case '+':
res = op1 + op2;
break;
case '-':
```

```
res=op1-op2;
break;
case '*':
res=op1*op2;
 break;
case '/':
res = op1 / op2;
break;
case '^':
res = pow(op1, op2);
break;
default:
printf(" Invalid Character \n");
}
push(res);
}
}
printf("Result of above expression is: %d\n", pop());
}
 void push(int element)
{
++top;
s[top]=element;
}
int pop()
{
int elements;
elements=s[top];
--top;
return(elements);
}
```

OUTPUT


Program to Evaluate Postfix Expression.
Enter the postfix expression:
45+3*2+
Result of above expression is: 29

Process exited after 13.51 seconds with return value 34
Press any key to continue . . .

//14. BINARY SEARCH TREE SORT

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int info;
struct node *left;
struct node *right;
};
typedef struct node NODE;
 NODE *root=NULL;
 //NODE *p;

 void disp(struct node *ptr,int level)
{
int i;
if(ptr!=NULL)
{
disp(ptr->right, level+1);
for(i=0;i<level;i++)
printf(" ");
printf("%2d\n", ptr->info);
disp(ptr->left, level+1);
}
}

void create (int item)
{
NODE *newnode, *currptr, *ptr;
newnode =(NODE *)malloc (sizeof (NODE));
newnode -> info = item;
```

```
newnode -> left = NULL;
newnode -> right = NULL;
if (root== NULL)
root = newnode;
else
{
currptr = root;
while (currptr != NULL)
{
        ptr=currptr;
        currptr = (item > currptr -> info)? currptr -> right: currptr -> left;
}
if (item< ptr -> info)
ptr -> left = newnode;
else
ptr -> right = newnode;
}

}
```

```
 NODE *getInSuccessor(NODE *ptr)
{
while(ptr->left != NULL)
ptr = ptr->left;
return ptr;
}
 //this will give the minimum key ;
```

```
NODE *deletion (NODE *p, int item)
{
NODE *temp;
 if(item > p->info)
p->right = deletion (p->right, item);
else
if(item < p->info)
p->left= deletion (p->left, item); /* executing else means got the key */
else
{
 if(p->left == NULL)
  {
temp = p->right;
free(p);
return temp;
}
else
if (p->right == NULL)
{
            temp =p->left;
free(p);
return temp;
}
temp = getInSuccessor (p->right);
p->info =temp->info;
p->right=deletion(p->right, temp->info);
}
return p;
}

int main()
```

```
{
int item, ch,n,i;
while(1)
{
printf("\n Binary Search Tree Menu");
printf("\n--------");
printf("\n 1. Insert ");
printf("\n 2. Delete ");
printf("\n 3. Display ");
printf("\n 4. Exit ");
printf("\n Enter the choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1:
printf("\n Enter the Number of Nodes :");
scanf("%d", &n);
for (i=0;i<n;i++)
{
printf("\n Enter the data for the node");
scanf("%d",&item);
create(item);
}
break;
case 2:
        printf("\n Enter an Item to be deleted: ");
        scanf("%d", &item);
root=deletion(root, item);
disp(root,1);
break;
case 3:
```

```
printf("\n The Binary Tree nodes are : \n\n\n\n");
disp(root,1);
break;
case 4 :
exit(1);
}
}
}
```

OUTPUT:

```
 Binary Search Tree Menu
--------
 1. Insert
 2. Delete
 3. Display
 4. Exit
 Enter the choice:
1

 Enter the Number of Nodes :5

 Enter the data for the node56

 Enter the data for the node78

 Enter the data for the node34

 Enter the data for the node24

 Enter the data for the node90

 Binary Search Tree Menu
--------
 1. Insert
 2. Delete
 3. Display
 4. Exit
 Enter the choice: 3

 The Binary Tree nodes are :


    90
  78
 56
  34
    24
```

```
 Binary Search Tree Menu
--------
 1. Insert
 2. Delete
 3. Display
 4. Exit
 Enter the choice: 3

 The Binary Tree nodes are :



   90
  78
 56
  34
   24

 Binary Search Tree Menu
--------
 1. Insert
 2. Delete
 3. Display
 4. Exit
 Enter the choice: 2

 Enter an Item to be deleted: 34
   90
  78
 56
  24
```

```
 Binary Search Tree Menu
--------
 1. Insert
 2. Delete
 3. Display
 4. Exit
 Enter the choice: 4

--------------------------------
Process exited after 68.43 seconds with return value 1
Press any key to continue . . .
```

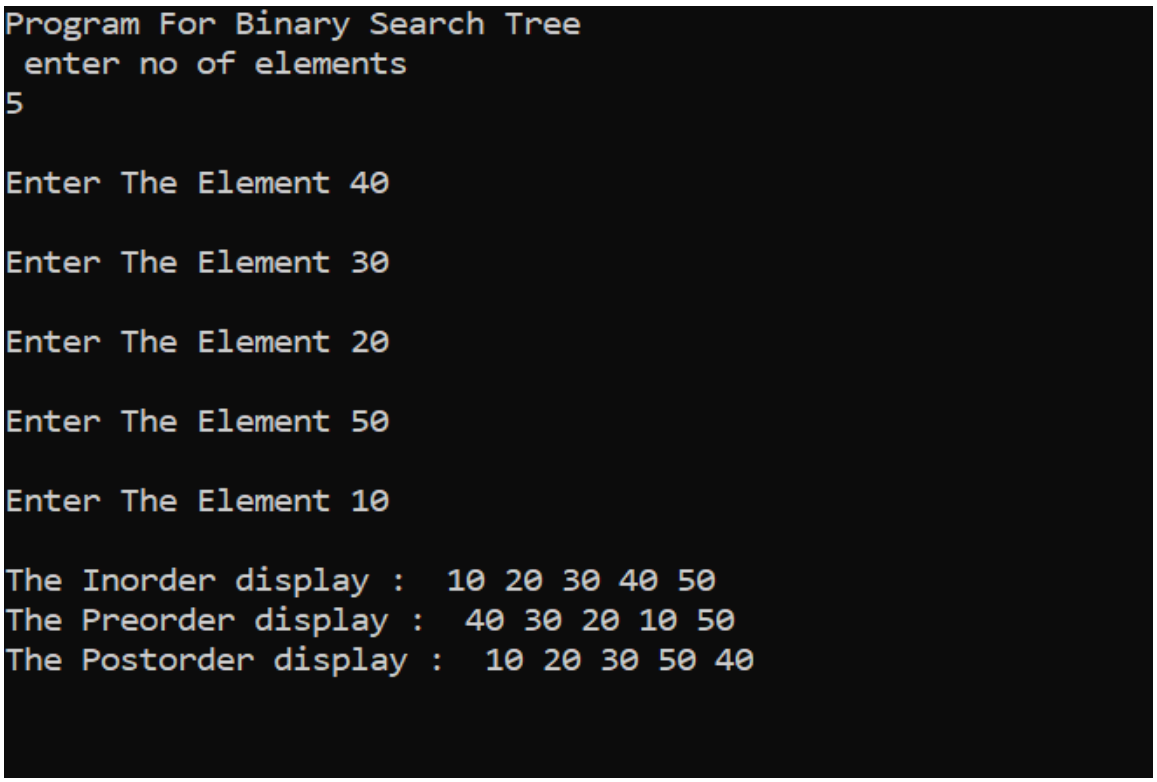
/*15.Write a program to create binary search tree and perform inorder, preorder and post order traversal. */

```
# include <stdio.h>

# include <stdio.h>

# include <conio.h>

# include <stdlib.h>

typedef struct BST {

int data;

struct BST *lchild, *rchild;

} node;

node *create_node() {

node *temp;

temp = (node *) malloc(sizeof(node));

temp->lchild = NULL;

temp->rchild = NULL;

return temp;

}

void insert(node *root, node *new_node) {

if (new_node->data < root->data) {

if (root->lchild == NULL)

root->lchild = new_node;

else

insert(root->lchild, new_node);

}

if (new_node->data > root->data) {

if (root->rchild == NULL)

root->rchild = new_node;

else

insert(root->rchild, new_node);

}

}

void inorder(node *temp) {
```

```
if (temp != NULL) {
inorder(temp->lchild);
printf("%3d", temp->data);
inorder(temp->rchild);
}
}
void preorder(node *temp) {
if (temp != NULL) {
printf("%3d", temp->data);
preorder(temp->lchild);
preorder(temp->rchild);
}
}
void postorder(node *temp) {
if (temp != NULL) {
postorder(temp->lchild);
postorder(temp->rchild);
printf("%3d", temp->data);
}
}
void main()
{
int n,i=1;
node *new_node, *root;
node *create_node();
root = NULL;
printf("\nProgram For Binary Search Tree\n ");
printf("enter no of elements\n");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
```

```
new_node = create_node();
printf("\nEnter The Element ");
scanf("%d", &new_node->data);
if (root == NULL) /* Tree is not Created */
root = new_node;
else
insert(root, new_node);
}
printf("\nThe Inorder display : ");
inorder(root);
printf("\nThe Preorder display : ");
preorder(root);
printf("\nThe Postorder display : ");
postorder(root);
getch();
}
```

OUTPUT



// Heap Sort

```
# include<stdio.h>
void heapify(int a[], int n, int i)
{
int largest = i;
int left = 2 * i + 1;
int right = 2 * i + 2;
if(left < n && a[left] > a[largest])
{
largest = left;
}
if(right < n && a[right] > a[largest])
{
largest = right;
}
if(largest !=i)
{
int temp;
temp = a[i];
a[i] = a[largest];
a[largest] = temp;
heapify(a,n,largest);
}
}
void HEAPSORT(int a[], int n)
{
int i;
for(i=n/2-1; i>=0;i--)
heapify(a,n,i);
for( i=n-1; i>=0; i--)
{
```

```
int temp;
temp = a[0];
a[0] = a[i];
a[i] = temp;
heapify(a,i,0);
}
}
void printArr(int arr[], int n)
{
int i;
for( i=0; i<n; ++i)
{
printf("%4d",arr[i]);
}
}
int main()
{
int a[20];
int m,i;
printf("enter the no of elements");
scanf("%d",&m);
printf("enter elements");
for(i=0;i<m;i++)
scanf("%d",&a[i]);
printf("\n Before sorting : ");
printArr(a,m);
HEAPSORT(a,m);
printf("\n After sorting : ");
printArr(a,m);
getch();
}
```

OUTPUT:

```
enter the no of elements
6
enter elements
56
7
90
35
100
79

 Before sorting :    56   7  90  35 100  79
 After sorting :     7  35  56  79  90 100
```

/* 17. Given S1={“Flowers”} ; S2={“are beautiful”} I. Find the length of S1 II. Concatenate S1 and S2 III. Extract the substring “low” from S1 IV. Find “are” in S2 and replace it with “is”*/

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
   char S1[] = "Flowers";
   char S2[] = "are beautiful";

   int choice;
   while(1)
   {
   printf("1. Find the length of S1\n");
   printf("2. Concatenate S1 and S2\n");
   printf("3. Extract the substring 'low' from S1\n");
   printf("4. Find 'are' in S2 and replace it with 'is'\n");
          printf("5.exit\n");
   printf("Enter your choice:\n");
   scanf("%d", &choice);
switch (choice) {
      case 1:
         printf("Length of S1: %zu\n", strlen(S1));
         break;
      case 2: {
         char result[100];
         strcpy(result, S1);
         strcat(result, " ");
         strcat(result, S2);
         printf("Concatenated string: %s\n", result);
         break;
```

```
        }
        case 3: {
            char substring[4];
            strncpy(substring, S1 + 1, 3);
            substring[3] = '\0';
            printf("Extracted substring from S1: %s\n", substring);
            break;
        }
        case 4: {



            char *found = strstr(S2, "are");
            if (found != NULL) {
                strncpy(found, "is ", 3);
                printf("Modified S2: %s\n", S2);
            } else {
                printf("'are' not found in S2\n");
            }
            break;
        }
        case 5:
             exit(0);
        default:
            printf("Invalid choice\n");
            break;
    }
}
    return 0;
}
```

OUTPUT

```
1. Find the length of S1
2. Concatenate S1 and S2
3. Extract the substring 'low' from S1
4. Find 'are' in S2 and replace it with 'is'
5.exit
Enter your choice:
1
Length of S1: 7
1. Find the length of S1
2. Concatenate S1 and S2
3. Extract the substring 'low' from S1
4. Find 'are' in S2 and replace it with 'is'
5.exit
Enter your choice:
2
Concatenated string: Flowers are beautiful
1. Find the length of S1
2. Concatenate S1 and S2
3. Extract the substring 'low' from S1
4. Find 'are' in S2 and replace it with 'is'
5.exit
Enter your choice:
3
Extracted substring from S1: low
1. Find the length of S1
2. Concatenate S1 and S2
3. Extract the substring 'low' from S1
4. Find 'are' in S2 and replace it with 'is'
5.exit
Enter your choice:
4
Modified S2: is  beautiful
1. Find the length of S1
2. Concatenate S1 and S2
3. Extract the substring 'low' from S1
4. Find 'are' in S2 and replace it with 'is'
5.exit
```