

EX.NO:1.A ARRAY IMPLEMENTATION OF STACK AND QUEUE ADTS**DATE:****AIM:**

To write a 'C' program for implementing stacks using array.

ALGORITHM

Step1: Define an array which stores stack elements.

Step 2: Increment Top pointer and then PUSH data into the stack at the position pointed by the Top.

Step 3: POP the data out the stack at the position pointed by the Top.

Step 5. The stack represented by array is traversed to display its content.

PROGRAM

```
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
    //clrscr();
    top=-1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d",&n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t ----- ");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
```

```

        display();
        break;
    }
    case 4:
    {
        printf("\n\t EXIT POINT ");
        break;
    }
    default:
    {
        printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
    }
}
}
while(choice!=4);
return 0;
}
void push()
{
    if(top>=n-1)
    {
        printf("\n\tSTACK is over flow");

    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}
void display()
{.
    if(top>=0)
    {

```

```

    printf("\n The elements in STACK \n");
    for(i=top; i>=0; i--)
        printf("\n%d",stack[i]);
    printf("\n Press Next Choice");
}
else
{
    printf("\n The STACK is empty");
}
}

```

OUTPUT

Enter the size of STACK[MAX=100]:10
 STACK OPERATIONS USING ARRAY

-
- 1.PUSH
 - 2.POP
 - 3.DISPLAY
 - 4.EXIT

Enter the Choice:1
 Enter a value to be pushed:12

Enter the Choice:1
 Enter a value to be pushed:24

Enter the Choice:1
 Enter a value to be pushed:98

Enter the Choice:3
 The elements in STACK
 98
 24
 12
 Press Next Choice
 Enter the Choice:2

The popped elements is 98
 Enter the Choice:3
 The elements in STACK
 24
 12
 Press Next Choice
 Enter the Choice:4
 EXIT POINT

RESULT:

Thus the program to implement stack using array has been implemented and the output is verified.

EX.NO:1.B**ARRAY IMPLEMENTATION OF QUEUE****DATE:****AIM:**

To write a 'C' program for implementing Queue using array.

ALGORITHM:

Step 1 :To insert Check if the queue is full or not.

Step 2 :If the queue is full, then print overflow error and exit the program.

Step 3 :If the queue is not full, then increment the tail and add the element.

Step 4 :Check if the queue is empty or not.

Step 5 :If the queue is empty, then print underflow error and exit the program.

Step 6 :If the queue is not empty, then print the element at the head and increment the head.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define n 5
void main()
{
    int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;
    printf("Queue using Array");
    printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");
    while(ch)
    {
        printf("\nEnter the Choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                if(rear==x)
                    printf("\n Queue is Full");
                else
                {
                    printf("\n Enter no %d:",j++);
                    scanf("%d",&queue[rear++]);
                }
                break;
            case 2:
                if(front==rear)
                {
                    printf("\n Queue is empty");
                }
                else
                {
                    printf("\n Deleted Element is %d",queue[front++]);
                    x++;
                }
                break;
            case 3:
```

```

printf("\n Queue Elements are:\n ");
if(front==rear)
    printf("\n Queue is Empty");
else
{
    for(i=front; i<rear; i++)
    {
        printf("%d",queue[i]);
        printf("\n");
    }
    break;
case 4:
    exit(0);
default:
    printf("Wrong Choice: please see the options");
}
}
}
}

```

OUTPUT:

Queue using Array

1.Insertion

2.Deletion

3.Display

4.Exit

Enter the Choice:1

Enter no 1:10

Enter the Choice:1

Enter no 2:54

Enter the Choice:1

Enter no 3:98

Enter the Choice:1

Enter no 4:234

Enter the Choice:3

Queue Elements are:

10

54

98

234

Enter the Choice:2

Deleted Element is 10

Enter the Choice:3

Queue Elements are:

54

98

234

Enter the Choice:4

RESULT:

Thus the 'C' program to implement Queue using array has been executed and the output is verified.

EX.NO:2**ARRAY IMPLEMENTATION OF LIST ADT****DATE:****AIM:**

To write a 'C' program for implementing List ADT using array.

ALGORITHM:

Step 1: Define an array to store the elements in the List

Step 2: Perform Insertion, Deletion operation in the List

Step 3: Search the obtained element in the List

Step 4: Traverse the list to display the elements.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
```

```
void create();
void insert();
void deletion();
void search();
void display();
int a,b[20], n, p, e, f, i, pos;
```

```
void main()
{
//clrscr();
int ch;
char g='y';

do
{
printf("\n main Menu");
printf("\n 1.Create \n 2.Delete \n 3.Search \n 4.Insert \n 5.Display\n 6.Exit \n");
printf("\n Enter your Choice");
scanf("%d", &ch);
```

```
switch(ch)
{
case 1:
create();
break;

case 2:
deletion();
break;
```

```

case 3:
search();
break;
case 4:
insert();
break;

case 5:
display();
break;

case 6:
exit();
break;
default:
printf("\n Enter the correct choice:");
}
printf("\n Do u want to continue::");
scanf("\n%c", &g);
}
while(g=='y'||g=='Y');
getch();
}

void create()
{
printf("\n Enter the number of nodes");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\n Enter the Element:",i+1);
scanf("%d", &b[i]);
}

}
void deletion()
{
printf("\n Enter the position u want to delete::");
scanf("%d", &pos);
if(pos>=n)
{
printf("\n Invalid Location::");
}
else
{
for(i=pos+1;i<n;i++)
{
b[i-1]=b[i];

```

```

}
n--;
}
printf("\n The Elements after deletion");
for(i=0;i<n;i++)

{
printf("\t%d", b[i]);
}
}

void search()
{
printf("\n Enter the Element to be searched:");
scanf("%d", &e);

for(i=0;i<n;i++)
{
if(b[i]==e)
{
printf("Value is in the %d Position", i);
}
else
{
printf("Value %d is not in the list::", e);
continue;
}
}
}

void insert()
{
printf("\n Enter the position u need to insert::");
scanf("%d", &pos);

if(pos>=n)
{
printf("\n invalid Location::");
}
else
{
for(i=MAX-1;i>=pos-1;i--)
{
b[i+1]=b[i];
}
printf("\n Enter the element to insert::\n");
scanf("%d",&p);
b[pos]=p;

```



```

n++;
}
printf("\n The list after insertion::\n");

display();
}

void display()
{
printf("\n The Elements of The list ADT are:");
for(i=0;i<n;i++)
{
printf("\n\n%d", b[i]);
}
}

```

OUTPUT:

```

2.Delete
3.Search
4.Insert
5.Display
6.Exit

Enter your Choice1
Enter the number of nodes2
Enter the Element:11
Enter the Element:23
Do u want to continue:::y

main Menu
1.Create
2.Delete
3.Search
4.Insert
5.Display

```

RESULT:

Thus the C program to implement List ADT using array has been executed and the output is verified.

EX.NO:3A LINKED LIST IMPLEMENTATION OF LIST(SINGLY LINKED LIST)
DATE:

AIM:

To write a 'C' program to create a singly linked list implementation.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the choice from the user.

Step3: If the choice is to add records, get the data from the user and add them to the list.

Step 4: If the choice is to delete records, get the data to be deleted and delete it from the list.

Step 5: If the choice is to display number of records, count the items in the list and display.

Step 6: If the choice is to search for an item, get the item to be searched and respond yes if the item is found, otherwise no.

Step 7: Terminate the program

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#define NULL 0
struct info
{
    int data;
    struct info *next;
};
struct info *head,*temp,*disp;
void additem();
void delitem();
void display();
int size();
void search();
void main()
{
    int choice;
    clrscr();
    while(1)
    {
        printf("\n1.Add records");
        printf("\n2.Delete records");
        printf("\n3.Display records");
        printf("\n4.Count no. of items in the list");
        printf("\n5.Searching an item in the list");
        printf("\n6.Exit");
        printf("\nEnter your choice:");
        scanf("%d",&choice);
        fflush(stdin);
        switch(choice)
```

```

{
case 1:
    additem();
    break;
case 2:
    delitem();
    break;
case 3:
    display();
    break;
case 4:
    printf("\nThe size of the list is %d",size());
    break;
case 5:
    search();
    break;
case 6:
    exit(0);
}
}
}
void additem()
{
struct info *add;
char proceed='y';
while(toupper(proceed)=='Y')
{
add=(struct info*)malloc(sizeof(struct info));
printf("Enter data:");
scanf("%d",&add->data);
fflush(stdin);
if(head==NULL)
{
head=add;
add->next=NULL;
temp=add;
}
else
{
temp->next=add;
add->next=NULL;
temp=add;
}
printf("\nWant to proceed y/n");
proceed=getchar();
fflush(stdin);
}
}

```

```

void delitem()
{
    struct info *curr,*prev;

    int tdata;
    if(head==NULL)
    {
        printf("\nNo records to delete");
        return;
    }
    printf("\nEnter the data to delete");
    scanf("%d",&tdata);
    fflush(stdin);
    prev=curr=head;
    while((curr!=NULL)&&(curr->data!=tdata))
    {
        prev=curr;
        curr=curr->next;
    }
    if(curr==NULL)
    {
        printf("\nData not found");
        return;
    }
    if(curr==head)
        head=head->next;
    else
    {
        /*for inbetween element deletion*/
        prev->next=curr->next;
        /*for the last element deletion*/
        if(curr->next==NULL)
            temp=prev;
    }
    free(curr);
}

void display()
{
    if(head==NULL)
    {
        printf("\nNo data to display");
        return;
    }
    for(dis=head;dis!=NULL;dis=dis->next)
    {
        printf("Data->%d",dis->data);
    }
}

```

```

int size()
{
int count=0;
if(head==NULL)
    return count;
for(dis=head;disp!=NULL;disp=disp->next)

    count++;

return count;
}
void search()
{
int titem,found=0;
if(head==NULL)
{
printf("\nNo data in the list");
return;
}
printf("\nEnter the no. to search:");
scanf("%d",&titem);
for(dis=head;disp!=NULL&&found==0;disp=disp->next)
{
if(disp->data==titem)
    found=1;
}
if(found==0)
    printf("\nSearch no. is not present in the list");
else
    printf("\nSearch no. is present in the list");
return;
}

```

OUTPUT:

```

1.Add records
2.Delete records
3.Display records
4.Count no. of items in the list
5.Searching an item in the list
6.Exit
Enter your choice:1
Enter data:12
Want to proceed y/ny
Enter data:13
Want to proceed y/ny
Enter data:41
Want to proceed y/nn
1.Add records
2.Delete records

```

3.Display records
 4.Count no. of items in the list
 5.Searching an item in the list
 6.Exit
 Enter your choice:3
 Data->12Data->13Data->41
 1.Add records
 2.Delete records
 3.Display records
 4.Count no. of items in the list
 5.Searching an item in the list
 6.Exit
 Enter your choice:4
 The size of the list is 3
 1.Add records
 2.Delete records
 3.Display records
 4.Count no. of items in the list
 5.Searching an item in the list
 6.Exit
 Enter your choice:2
 Enter the data to delete13
 1.Add records
 2.Delete records
 3.Display records
 4.Count no. of items in the list
 5.Searching an item in the list
 6.Exit
 Enter your choice:3
 Data->12Data->41
 1.Add records
 2.Delete records
 3.Display records
 4.Count no. of items in the list
 5.Searching an item in the list
 6.Exit
 Enter your choice:5
 Enter the no. to search:13
 Search no. is not present in the list
 1.Add records
 2.Delete records
 3.Display records
 4.Count no. of items in the list
 5.Searching an item in the list
 6.Exit
 Enter your choice:6

RESULT:

The given program is implemented, executed, tested and verified successfully.

EXNO:3B LINKED LIST IMPLEMENTATION OF LIST (DOUBLY LINKED LIST)**DATE:****AIM:**

To write a 'C' program to create a Doubly linked list implementation.

ALGORITHM:

Step 1:Start the program.

Step 2:Get the choice from the user.

Step 3:If the choice is to add records, get the data from the user and add them to the list.

Step 4:If the choice is to delete records, get the data to be deleted and delete it from the list.

Step 5:If the choice is to display number of records, count the items in the list and display.

Step 6:If the choice is to search for an item, get the item to be searched and respond yes if the item is found, otherwise no.

Step 7:Terminate the program

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#define NULL 0
struct info
{
int data;
struct info *next;
struct info *prev;
};
struct info *head,*temp,*disp;
void additem();
void delitem();
void display();
int size();
void search();
void main()
{
int choice;
clrscr();
while(1)
{
printf("\n1.Add records");
printf("\n2.Delete records");
printf("\n3.Display records");
printf("\n4.Count no. of items in the list");
printf("\n5.Searching an item in the list");
printf("\n6.Exit");
printf("\nEnter your choice:");
scanf("%d",&choice);
fflush(stdin);
```

```

switch(choice)
{

case 1:
    additem();
    break;
case 2:
    delitem();
    break;
case 3:
    display();
    break;
case 4:
    printf("\nThe size of the list is %d",size());
    break;
case 5:
    search();
    break;
case 6:
    exit(0);
}
}
}
void additem()
{
    struct info *add;
    char proceed='y';
    while(toupper(proceed)=='Y')
    {
        add=(struct info*)malloc(sizeof(struct info));
        printf("Enter data:");
        scanf("%d",&add->data);
        fflush(stdin);
        if(head==NULL)
        {
            head=add;
            add->next=NULL;
            add->prev=NULL;
            temp=add;
        }
        else
        {
            temp->next=add;
            add->prev=temp;
            add->next=NULL;
            temp=add;
        }
        printf("\nWant to proceed y/n");
    }
}

```



```

proceed=getchar();
fflush(stdin);
}
}
void delitem()
{
int x;
struct info *p;;
if(head==NULL)
{
printf("\nNo items in the list");
return;
}
printf("\nEnter the data to delete");
scanf("%d",&x);
//fflush(stdin);
p=(struct info *)malloc(sizeof(struct info));
p=head->next;
if(head->data==x)
{
head=head->next;
return;
}
while(p)
{
if(p->data==x)
{
p->prev->next=p->next;
if(p->next!=NULL)
p->next->prev=p->prev;
else
temp=p->prev;
return;
}
else
{
p=p->next;
}
}
printf("\nInvalid input");
}
void display()
{
if(head==NULL)
{
printf("\nNo data to display");
return;
}

```

```

printf("\nFrom forward direction\n");
for(dis=head;dis!=NULL;dis=dis->next)
{
printf("Data->%d",dis->data);
}
printf("\nFrom backward direction\n");
for(dis=temp;dis!=NULL;dis=dis->prev)

{
    printf("Data->%d",dis->data);
}
}
int size()
{
int count=0;
if(head==NULL)
    return count;
for(dis=head;dis!=NULL;dis=dis->next)
    count++;
return count;
}
void search()
{
int titem,found=0;
if(head==NULL)
{
printf("\nNo data in the list");
return;
}
printf("\nEnter the no. to search:");
scanf("%d",&titem);
for(dis=head;dis!=NULL&&found==0;dis=dis->next)
{
if(dis->data==titem)
    found=1;
}
if(found==0)
    printf("\nSearch no. is not present in the list");
else
    printf("\nSearch no. is present in the list");
return;
}

```

OUTPUT:

- 1.Add records
- 2.Delete records
- 3.Display records

4.Count no. of items in the list
 5.Searching an item in the list
 6.Exit

Enter your choice:1

Enter data:21

Want to proceed y/ny

Enter data:23

Want to proceed y/ny

Enter data:45

Want to proceed y/nn

1.Add records
 2.Delete records
 3.Display records
 4.Count no. of items in the list
 5.Searching an item in the list
 6.Exit

Enter your choice:3

From forward direction

Data->21Data->23Data->45

From backward direction

Data->45Data->23Data->21

1.Add records
 2.Delete records
 3.Display records
 4.Count no. of items in the list
 5.Searching an item in the list
 6.Exit

Enter your choice:2 Enter
 the data to delete23

1.Add records
 2.Delete records
 3.Display records
 4.Count no. of items in the list
 5.Searching an item in the list
 6.Exit

Enter your choice:4

The size of the list is 2

1.Add records
 2.Delete records
 3.Display records
 4.Count no. of items in the list

5.Searching an item in the list
6.Exit
Enter your choice:3
From forward direction
Data->21Data->45
From backward direction
Data->45Data->21
1.Add records
2.Delete records
3.Display records
4.Count no. of items in the list
5.Searching an item in the list
6.Exit
Enter your choice:5 Enter
the no. to search:45

Search no. is present in the list
1.Add records
2.Delete records
3.Display records
4.Count no. of items in the list
5.Searching an item in the list
6.Exit
Enter your choice:6

RESULT:

The given program is implemented, executed, tested and verified successfully.

EXNO:3C**LINKED LIST IMPLEMENTATION OF STACK****DATE:****AIM:**

To write a 'C' program to create a Stack linked list implementation.

ALGORITHM:

push(value) - Inserting an element into the Stack

We can use the following s to insert a new node into the stack...

- 1: Create a newNode with given value.
- 2: Check whether stack is Empty (top == NULL)
- 3: If it is Empty, then set newNode → next = NULL.
- 4: If it is Not Empty, then set newNode → next = top.
- 5: Finally, set top = newNode.

pop() - Deleting an Element from a Stack

We can use the following s to delete a node from the stack...

- 1: Check whether stack is Empty (top == NULL).
- 2: If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function
- 3: If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
- 4: Then set 'top = top → next'.
- 7: Finally, delete 'temp' (free(temp)).

display() - Displaying stack of elements

We can use the following s to display the elements (nodes) of a stack...

- 1: Check whether stack is Empty (top == NULL).
- 2: If it is Empty, then display 'Stack is Empty!!!' and terminate the function.
- 3: If it is Not Empty, then define a Node pointer 'temp' and initialize with top.
- 4: Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack (temp → next != NULL).
- 4: Finally! Display 'temp → data ---> NULL'.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct Node
{
    int data;
    struct Node *next;
}*top = NULL;
void push(int);
void pop();
void display();
void main()
{
    int choice, value;
    clrscr();
    printf("\n:: Stack using Linked List ::\n");
    while(1){
        printf("\n***** MENU *****\n");
```

```

printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
printf("Enter your choice: ");
scanf("%d",&choice);
switch(choice){
    case 1: printf("Enter the value to be insert: ");
            scanf("%d", &value);
            push(value);
            break;
    case 2: pop(); break;
    case 3: display(); break;
    case 4: exit(0);
    default: printf("\nWrong selection!!! Please try again!!!\n");
}
}
}
void push(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    if(top == NULL)
        newNode->next = NULL;
    else
        newNode->next = top;
    top = newNode;
    printf("\nInsertion is Success!!!\n");
}
void pop()
{
    if(top == NULL)
        printf("\nStack is Empty!!!\n");
    else{
        struct Node *temp = top;
        printf("\nDeleted element: %d", temp->data);
        top = temp->next;
        free(temp);
    }
}
void display()
{
    if(top == NULL)
        printf("\nStack is Empty!!!\n");
    else{
        struct Node *temp = top;
        while(temp->next != NULL){
            printf("%d--->",temp->data);
            temp = temp -> next;
        }
    }
}

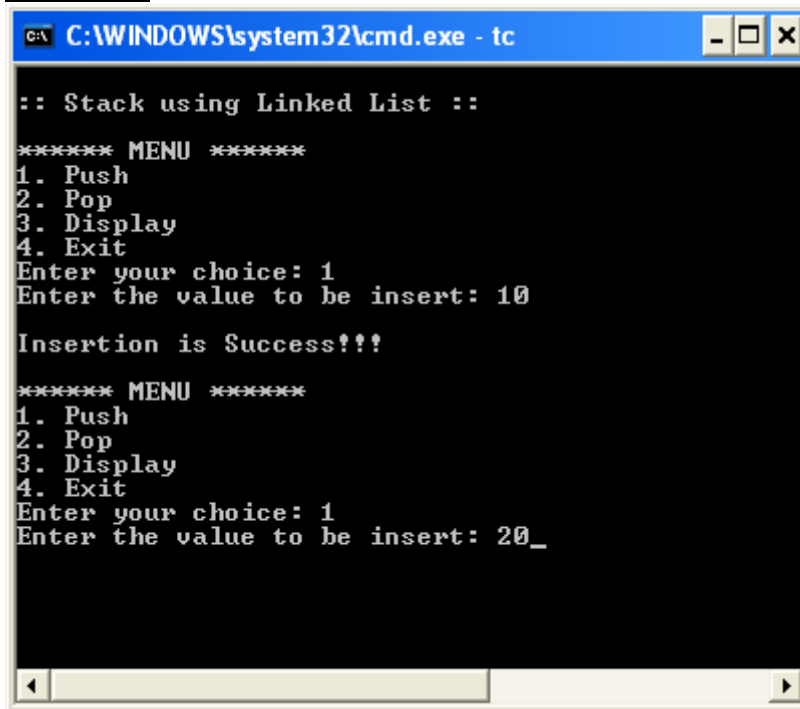
```

```

    printf("%d--->NULL",temp->data);
}
}

```

OUTPUT:



```

C:\WINDOWS\system32\cmd.exe - tc

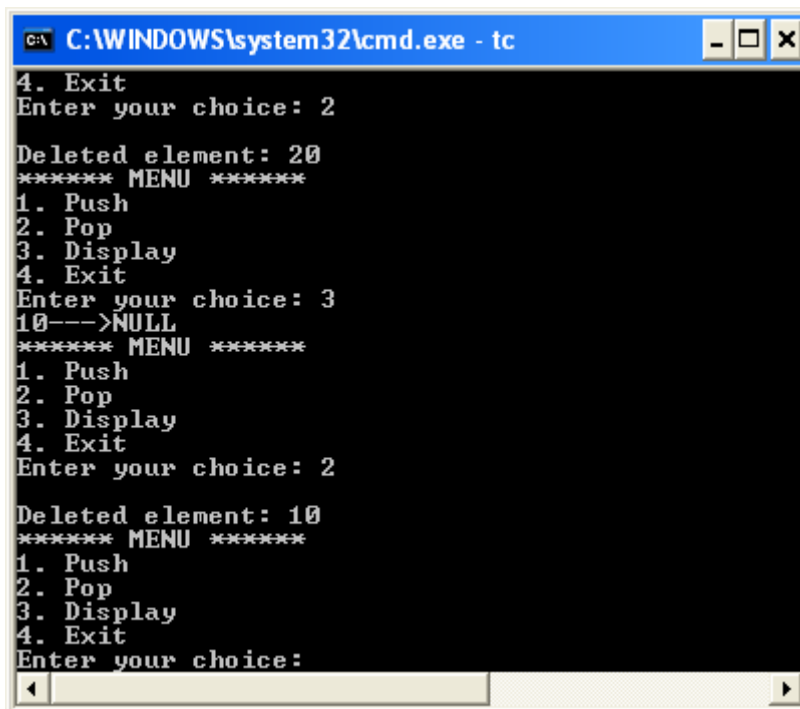
:: Stack using Linked List ::

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 10

Insertion is Success!!!

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20_

```



```

C:\WINDOWS\system32\cmd.exe - tc

4. Exit
Enter your choice: 2

Deleted element: 20
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
10--->NULL
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2

Deleted element: 10
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:

```

RESULT:

The given program is implemented, executed, tested and verified successfully.

EXNO:3D**LINKED LIST IMPLEMENTATION OF QUEUE****DATE:****AIM:**

To write a 'C' program to create a Stack linked list implementation.

ALGORITHM:

Queue using Linked List

enQueue(value) - Inserting an element into the Queue

We can use the following s to insert a new node into the queue...

- 1: Create a newNode with given value and set 'newNode → next' to NULL.
- 2: Check whether queue is Empty (rear == NULL)
- 3: If it is Empty then, set front = newNode and rear = newNode.
- 4: If it is Not Empty then, set rear → next = newNode and rear = newNode.

deQueue() - Deleting an Element from Queue

We can use the following s to delete a node from the queue...

- 1: Check whether queue is Empty (front == NULL).
- 2: If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function
- 3: If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.
- 4: Then set 'front = front → next' and delete 'temp' (free(temp)).

display() - Displaying the elements of Queue

We can use the following s to display the elements (nodes) of a queue...

- 1: Check whether queue is Empty (front == NULL).
- 2: If it is Empty then, display 'Queue is Empty!!!' and terminate the function.
- 3: If it is Not Empty then, define a Node pointer 'temp' and initialize with front.
- 4: Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL).
- 4: Finally! Display 'temp → data ---> NULL'

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct Node
{
    int data;
    struct Node *next;
}*front = NULL,*rear = NULL;
void insert(int);
void delete();
void display();
void main()
{
    int choice, value;
    clrscr();
    printf("\n:: Queue Implementation using Linked List ::\n");
    while(1){
        printf("\n***** MENU *****\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
```



```

scanf("%d",&choice);
switch(choice){
case 1: printf("Enter the value to be insert: ");
        scanf("%d", &value);
        insert(value);
        break;
case 2: delete(); break;
case 3: display(); break;
case 4: exit(0);
default: printf("\nWrong selection!!! Please try again!!!\n");
}
}
}
void insert(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode -> next = NULL;
    if(front == NULL)
        front = rear = newNode;
    else{
        rear -> next = newNode;
        rear = newNode;
    }
    printf("\nInsertion is Success!!!\n");
}
void delete()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else{
        struct Node *temp = front;
        front = front -> next;
        printf("\nDeleted element: %d\n", temp->data);
        free(temp);
    }
}
void display()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else{
        struct Node *temp = front;
        while(temp->next != NULL){
            printf("%d--->",temp->data);
            temp = temp -> next;
        }
    }
}

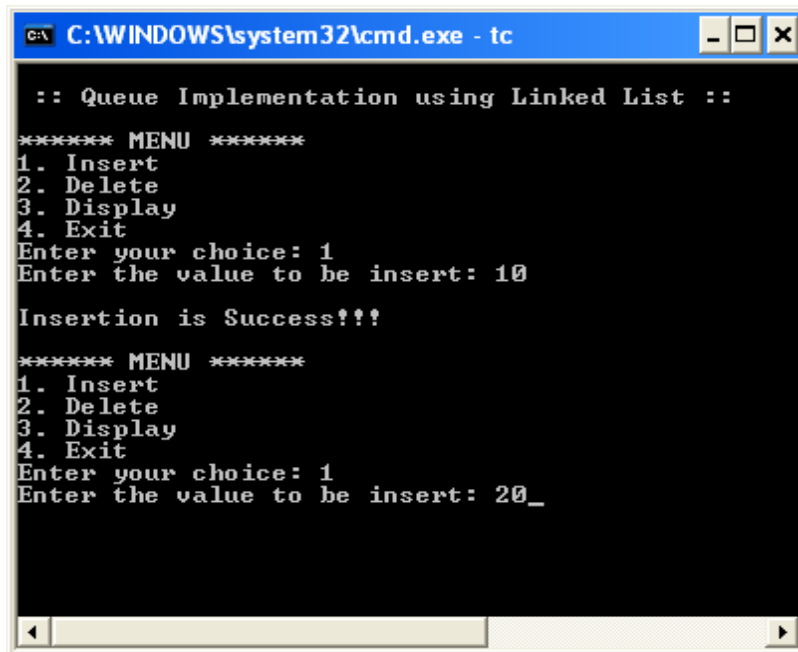
```

```

printf("%d--->NULL\n",temp->data);
}
}

```

OUTPUT:



```

C:\WINDOWS\system32\cmd.exe - tc

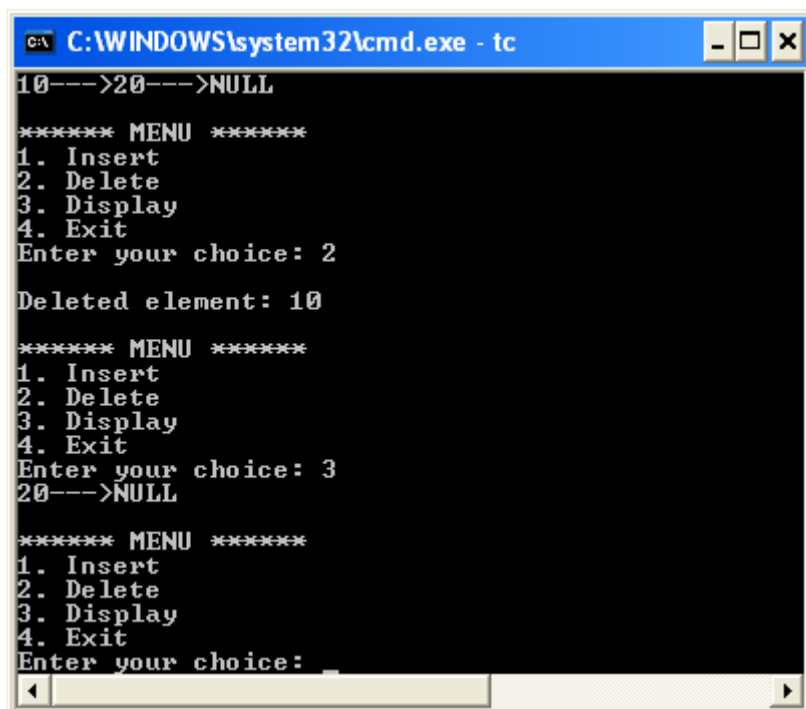
:: Queue Implementation using Linked List ::

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 10

Insertion is Success!!!

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20_

```



```

C:\WINDOWS\system32\cmd.exe - tc

10--->20--->NULL

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

Deleted element: 10

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
20--->NULL

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: _

```

RESULT:

The given program is implemented, executed, tested and verified successfully.

EX.NO:4A**APPLICATIONS OF LIST-POLYNOMIAL ADDITION****DATE:****AIM:**

To write a 'C' program to represent a polynomial as a linked list and write functions for polynomial addition

ALGORITHM:

Step1:Start the program

Step2:Get the coefficients and powers for the two polynomials to be added.

Step3:Add the coefficients of the respective powers.

Step4:Display the added polynomial.

Step5:Terminate the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct polynomial
{
    int coff;
    int pow;
    struct polynomial *link;
} *ptr,*start1,*node,*start2,*start3,*ptr1,*ptr2;
typedef struct polynomial pnl;
int temp1,temp2;

void main()
{
    void create(void);
    void prnt(void);
    void suml(void);
    void sort(void);
    clrscr();
    printf("Enrter the elements of the first polynomial :");
    node = (pnl *) malloc(sizeof (pnl));
    start1=node;
    if (start1==NULL)
    {
        printf(" Unable to create memory.");
        getch();
        exit();
    }
    create();
    printf("Enter the elements of the second poly :");
    node = (pnl *) malloc(sizeof (pnl));
```

```

start2=node;
if (start2==NULL)
{
    printf("Unable to create memory.");
    getch();
    exit();
}
create();
clrscr();
//printing the elements of the lists
printf("The elements of the poly first are :");
ptr=start1;
prnt();
printf("The elements of the poly second are :");
ptr=start2;
prnt();
printf("The first sorted list is :");
ptr=start1;
sort();
ptr=start1;
prnt();
printf("The second sorted list is :");
ptr=start2;
sort();
ptr=start2;
prnt();
printf("The sum of the two lists are :");
suml();
ptr=start3;
prnt();
getch();
}
/*-----*/
void create()
{
    char ch;
    while(1)
    {
        printf(" Enter the coff and pow :");
        scanf("%d%d",&node->coff,&node->pow);
        if (node->pow==0 )
        {
            ptr=node;
            node=(pnl *)malloc(sizeof(pnl));
            node=NULL;
            ptr->link=node;
            break;
        }
    }
}

```

```

printf("Do u want enter more coff?(y/n)");
fflush(stdin);
scanf("%c",&ch);
if (ch=='n' )
{
    ptr=node;
    node=(pnl *)malloc(sizeof(pnl));
    node=NULL;
    ptr->link=node;
    break;
}
ptr=node;
node=(pnl *)malloc(sizeof(pnl));
ptr->link=node;
}
}
/*-----*/
void prnt()
{
    int i=1;
    while(ptr!=NULL )
    {
        if(i!=1)
            printf("+ ");
        printf(" %dx^%d\n ",ptr->coff,ptr->pow);
        ptr=ptr->link;
        i++;
    }
    //printf(" %d^%d",ptr->coff,ptr->pow);
}
/*-----*/
void sort()
{
    for(;ptr->coff!=NULL;ptr=ptr->link)
    for(ptr2=ptr->link;ptr2->coff!=NULL;ptr2=ptr2->link)
    {
        if(ptr->pow>ptr2->pow)
        {
            temp1=ptr->coff;
            temp2=ptr->pow;
            ptr->coff=ptr2->coff;
            ptr->pow=ptr2->pow;
            ptr2->coff=temp1;
            ptr2->pow=temp2;
        }
    }
}
/*-----*/

```

```

void suml()
{
node=(pnl *)malloc (sizeof(pnl));
start3=node;

ptr1=start1;
ptr2=start2;

while(ptr1!=NULL && ptr2!=NULL)
{
ptr=node;
if (ptr1->pow > ptr2->pow )
{
node->coff=ptr2->coff;
node->pow=ptr2->pow;
ptr2=ptr2->link; //update ptr list B
}
else if ( ptr1->pow < ptr2->pow )
{
node->coff=ptr1->coff;
node->pow=ptr1->pow;
ptr1=ptr1->link; //update ptr list A
}
else
{
node->coff=ptr2->coff+ptr1->coff;
node->pow=ptr2->pow;
ptr1=ptr1->link; //update ptr list A
ptr2=ptr2->link; //update ptr list B
}

node=(pnl *)malloc (sizeof(pnl));
ptr->link=node; //update ptr list C
} //end of while

if (ptr1==NULL) //end of list A
{
while(ptr2!=NULL)
{
node->coff=ptr2->coff;
node->pow=ptr2->pow;
ptr2=ptr2->link; //update ptr list B
ptr=node;
node=(pnl *)malloc (sizeof(pnl));
ptr->link=node; //update ptr list C
}
}
else if (ptr2==NULL) //end of list B

```

```

{
while(ptr1!=NULL)
{
node->coff=ptr1->coff;
node->pow=ptr1->pow;
ptr1=ptr1->link; //update ptr list B
ptr=node;
node=(pnl *)malloc (sizeof(pnl));
ptr->link=node; //update ptr list C
}
}
node=NULL;
ptr->link=node;
}

```

OUTPUT:

Enter the elements of the first polynomial : Enter the coff and pow :1 1
Do u want enter more coff ?(y/n)y
Enter the coff and pow :1 0
Enter the elements of the second poly : Enter the coff and pow :1 1
Do u want enter more coff ?(y/n)y
Enter the coff and pow :2 0
The elements of the poly first are : $1x^1 + 1x^0$
The elements of the poly second are : $1x^1 + 2x^0$
The first sorted list is : $1x^0 + 1x^1$
The second sorted list is : $2x^0 + 1x^1$
The sum of the two lists are : $3x^0 + 2x^1$

RESULT:

The given program is implemented, executed, tested and verified successfully

EX.NO:4B APPLICATIONS OF STACK- CONVERT INFIX TO POSTFIX EXPRESSION**DATE:****AIM:**

To write a 'C' program to implement stack and use it to convert infix to postfix expression.

ALGORITHM:

1. Start the program
2. Scan the Infix string from left to right.
3. Initialise an empty stack.
4. If the scanned character is an operand, add it to the Postfix string. If the scanned character is an operator and if the stack is empty Push the character to stack.
5. If the scanned character is an Operand and the stack is not empty, compare the precedence of the character with the element on top of the stack (topStack). If topStack has higher precedence over the scanned character Pop the stack else Push the scanned character to stack. Repeat this step as long as stack is not empty and topStack has precedence over the character.
6. Repeat this step till all the characters are scanned.
7. (After all characters are scanned, we have to add any character that the stack may have to the Postfix string.) If stack is not empty add topStack to Postfix string and Pop the stack. Repeat this step as long as stack is not empty.
8. Return the Postfix string.
9. Terminate the program.

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>
char stack[100];
int top=0;
char exp[100];
struct table
{
    char s[2];
    int isp;
    int icp;
}pr[7];
int isp(char c)
{
    int i;
    for(i=0;i<=6;i++)
        if(pr[i].s[0]==c)
            return(pr[i].isp);
    return 0;
}
int icp(char c)
{

```



```

int i;
for(i=0;i<=6;i++)
if(pr[i].s[0]==c)
return(pr[i].icp);
return 0;
}
void main()
{
int i;
clrscr();
strcpy(pr[0].s,"^");
pr[0].isp=3;
pr[0].icp=4;

strcpy(pr[1].s,"*");
pr[1].isp=2;
pr[1].icp=2;

strcpy(pr[2].s,"/");
pr[2].isp=2;
pr[2].icp=2;
strcpy(pr[3].s,"+");
pr[3].isp=1;
pr[3].icp=1;

strcpy(pr[4].s,"-");
pr[4].isp=1;
pr[4].icp=1;

strcpy(pr[5].s,"(");
pr[5].isp=0;
pr[5].icp=4;

strcpy(pr[6].s,"=");
pr[6].isp=-1;
pr[6].icp=0;

clrscr();
stack[top]='=';
printf("enter the infix expression");
gets(exp);
i=0;
printf("the postfix expression is ")
while(i<strlen(exp))
{
if(isalpha(exp[i])==0)
{

```

```

if(exp[i]==')')
{
    while(stack[top]!='(')
    {
        printf("%c",stack[top]);
        top--;
    }
    top--;
}
else
{
    while(isp(stack[top])>=icp(exp[i]))
    {
        printf("%c",stack[top]);
        top--;
    }
    top++;
    stack[top]=exp[i];
}
}

else
printf("%c",exp[i]);
i++;
}
while(top>0)
{
    printf("%c",stack[top]);
    top--;
}
getch();
}

```

OUTPUT:

enter the infix expression a*(s+d/f)+c
the postfix expression is asdf/+*c+

RESULT:

The given program is implemented, executed, tested and verified successfully

EX.NO:5 IMPLEMENTATION OF BINARY TREES AND OPERATIONS OF BINARY TREES**AIM:**

To write a 'C' program to implement an expression tree. Produce its pre-order, in-order, and post-order traversals.

ALGORITHM:

Step 1: Start the process.

Step 2: Initialize and declare variables.

Step 3: Enter the choice. Inorder / Preorder / Postorder.

Step 4: If choice is Inorder then

- Traverse the left subtree in inorder.
- Process the root node.
- Traverse the right subtree in inorder.

Step 5: If choice is Preorder then

- Process the root node.
- Traverse the left subtree in preorder.
- Traverse the right subtree in preorder.

Step 6: If choice is postorder then

- Traverse the left subtree in postorder.
- Traverse the right subtree in postorder.
- Process the root node.

Step7: Print the Inorder / Preorder / Postorder traversal.

Step 8: Stop the process.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
typedef struct treenode
{
    int data;
    struct treenode *left;
    struct treenode *right;
}tnode;
tnode *insertion(int,tnode*);
void preorder(tnode *);
void inorder(tnode *);
void postorder(tnode *);
void main()
{
    tnode *T=NULL;
    int ch1,n;
    char ch2;
```

```

do
{
    clrscr();
    printf("\n\t\t****Operation With Tree****");
    printf("\n\t1.Insertion");
    printf("\n\t2.Inorder Traversal");
    printf("\n\t3.Preorder Traversal");
    printf("\n\t4.Postorder Traversal");
    printf("\n\tEnter Your Choice   :");
    scanf("%d",&ch1);
    switch(ch1)
    {
        case 1:
            printf("\n\ntenter the element to be inserted :");
            scanf("%d",&n);
            T=insertion(n,T);
            break;
        case 2:
            inorder(T);
            break;
        case 3:
            preorder(T);

            break;
        case 4:
            postorder(T);
            break;
        default:
            printf("\n\nInvalid Option");
            break;
    }
    printf("\n\nDo you want to continue y/n   : ");
    scanf("%s",&ch2);
} while(ch2=='y');
getch();
}

```

```

tnode *insertion(int x,tnode *T)
{
    if(T==NULL)
    {
        T=(tnode *)malloc(sizeof(tnode));
        if(T==NULL)
            printf("\nout of space");
        else
            {

```

```

        T->data=x;
        T->left=T->right=NULL;
    }
}
else
{
    if(x<(T->data))
        T->left=insertion(x,T->left);
    else
    {
        if(x>T->data)
            T->right=insertion(x,T->right);
    }
}
return T;
}

```

```

void preorder(tnode *T)
{
    if(T!=NULL)
    {
        printf("\t%d",T->data);
        preorder(T->left);
        preorder(T->right);
    }
}

```

```

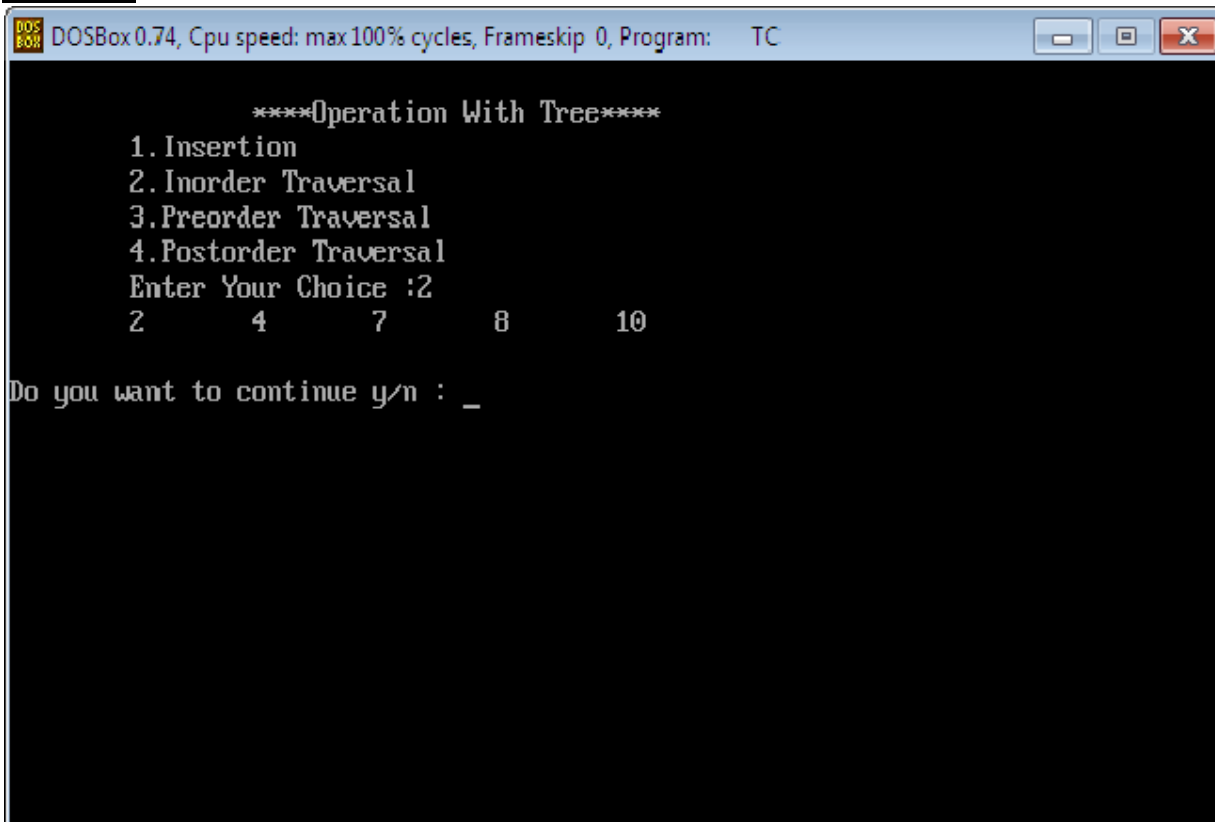
void postorder(tnode *T)
{
    if(T!=NULL)
    {
        postorder(T->left);
        postorder(T->right);
        printf("\t%d",T->data);
    }
}

```

```

void inorder(tnode *T)
{
    if(T!=NULL)
    {
        inorder(T->left);
        printf("\t%d",T->data);
        inorder(T->right);
    }
}

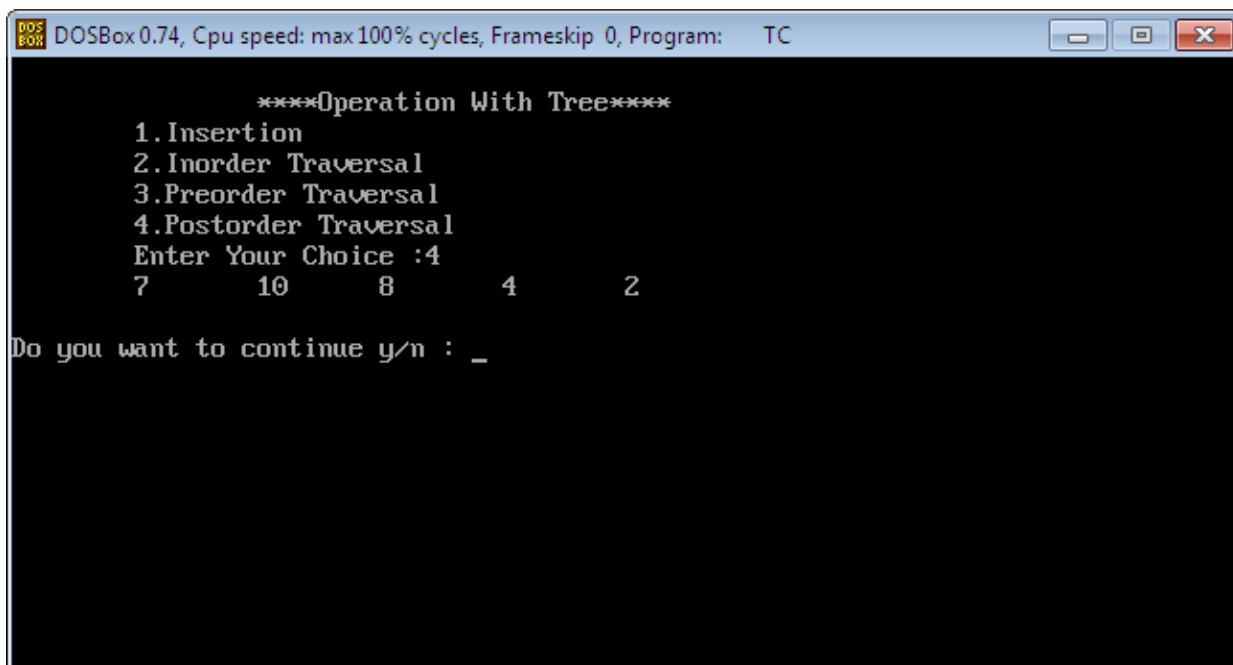
```

OUTPUT

```
DOSBox 0.74, Cpu speed: max100% cycles, Frameskip 0, Program: TC

      ****Operation With Tree****
1.Insertion
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
Enter Your Choice :2
2      4      7      8      10

Do you want to continue y/n : _
```



```
DOSBox 0.74, Cpu speed: max100% cycles, Frameskip 0, Program: TC

      ****Operation With Tree****
1.Insertion
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
Enter Your Choice :4
7      10     8      4      2

Do you want to continue y/n : _
```

RESULT:

The given program is implemented, executed, tested and verified successfully.

EX.NO:6**IMPLEMENT BINARY SEARCH TREE****DATE:****AIM:**

To write a 'C' program to implement binary search tree.

ALGORITHM:

Step 1: Start the process.

Step 2: Initialize and declare variables.

Step 3: Construct the Tree

Step 4: Data values are given which we call a key and a binary search tree

Step 5: To search for the key in the given binary search tree, start with the root node and Compare the key with the data value of the root node. If they match, return the root pointer.

Step 6: If the key is less than the data value of the root node, repeat the process by using the left subtree.

Step 7: Otherwise, repeat the same process with the right subtree until either a match is found or the subtree under consideration becomes an empty tree.

Step 8: Terminate

PROGRAM:

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

struct tree
{
    int data;
    struct tree *lchild;
    struct tree *rchild;
} *t,*temp;

int element;
void inorder(struct tree *);
void preorder(struct tree *);
void postorder(struct tree *);
struct tree * create(struct tree *, int);
struct tree * find(struct tree *, int);
struct tree * insert(struct tree *, int);
struct tree * del(struct tree *, int);
struct tree * findmin(struct tree *);
struct tree * findmax(struct tree *);
void main()
{
```


b
r
e
a
k
;

```

        case 7:
            inorder(t);
            break;
        case 8:
            preorder(t);
            break;
        case 9:
            postorder(t);
            break;
        case 10:
            exit(0);
    }
}while(ch<=10);
}

struct tree * create(struct tree *t, int element)
{
    t=(struct tree *)malloc(sizeof(struct tree));
    t->data=element;
    t->lchild=NULL;
    t->rchild=NULL;
    return t;
}

struct tree * find(struct tree *t, int element)
{
    if(t==NULL)
        return NULL;
    if(element<t->data)
        return(find(t->lchild,element));
    else
        if(element>t->data)
            return(find(t->rchild,element));
        else
            return t;
}

struct tree *findmin(struct tree *t)
{
    if(t==NULL)
        return NULL;
    else
        if(t->lchild==NULL)
            return t;
        else
            return(findmin(t->lchild));
}

```

```

struct tree *findmax(struct tree *t)
{
    if(t!=NULL)
    {
        while(t->rchild!=NULL)
            t=t->rchild;
    }
    return t;
}

```

```

struct tree *insert(struct tree *t,int element)
{
    if(t==NULL)
    {
        t=(struct tree *)malloc(sizeof(struct tree));
        t->data=element;
        t->lchild=NULL;
        t->rchild=NULL;
        return t;
    }
    else
    {
        if(element<t->data)
        {
            t->lchild=insert(t->lchild,element);
        }
        else
        {
            if(element>t->data)
            {
                t->rchild=insert(t->rchild,element);
            }
            else
            {
                if(element==t->data)
                {
                    printf("element already present\n");
                }
                return t;
            }
        }
    }
}

```

```

struct tree * del(struct tree *t, int element)
{
    if(t==NULL)
        printf("element not found\n");
    else
        if(element<t->data)
            t->lchild=del(t->lchild,element);

```

else

```

        if(element>t->data)
            t->rchild=del(t->rchild,element);
        else
            if(t->lchild&& t->rchild)
            {
                temp=findmin(t->rchild);
                t->data=temp->data;
                t->rchild=del(t->rchild,t->data);
            }
            else
            {
                temp=t;
                if(t->lchild==NULL)
                    t=t->rchild;
                else
                    if(t->rchild==NULL)
                        t=t->lchild;
                    free(temp);
            }

        return t;
    }

```

```

void inorder(struct tree *t)
{
    if(t==NULL)
        return;
    else
    {
        inorder(t->lchild);
        printf("\t%d",t->data);
        inorder(t->rchild);
    }
}

```

```

void preorder(struct tree *t)
{
    if(t==NULL)
        return;
    else
    {
        printf("\t%d",t->data);
        preorder(t->lchild);
        preorder(t->rchild);
    }
}

```

```
void postorder(struct tree *t)
{
```

```

    if(t==NULL)
        return;
    else
    {
        postorder(t->lchild);
        postorder(t->rchild);
        printf("\t%d",t->data);
    }
}

```

OUTPUT:

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :1

Enter the data:10

10

8.Preorder 9.Postorder

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder

BINARY
SEARCH
TREE

10.Exit

Enter ur choice :2

Enter the data:20

10 20

BINARY SEARCH TREE

Main Menu

1.Create

2.Insert

3.Delete

4.Find

5.FindMin

6.FindMax

7.Inorder

8.Preorder

9.Postorder

10.Exit

Enter ur choice :2

Enter the data:30

10 20 30

BINARY SEARCH TREE

Main Menu

1.Create

2.Insert

3.Delete

4.Find

5.FindMin

6.FindMax

7.Inorder

8.Preorder

9.Postorder

10.Exit

Enter ur choice :2

Enter the data:25

10 20 25 30

BINARY SEARCH TREE

Main Menu

1.Create
 2.Insert
 3.Delete
 4.Find
 5.FindMin
 6.FindMax
 7.Inorder
 8.Preorder
 9.Postorder
 10.Exit
 Enter ur choice :4

Enter the data:25

Element 25 is at 2216

BINARY SEARCH TREE

Main Menu

1.Create
 2.Insert
 3.Delete
 4.Find
 5.FindMin
 6.FindMax
 7.Inorder
 8.Preorder
 9.Postorder
 10.Exit
 Enter ur choice :5

Max element=10

BINARY SEARCH TREE

1.Create
 2.Insert
 3.Delete
 4.Find
 5.FindMin
 6.FindMax
 7.Inorder
 8.Preorder
 9.Postorder

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder

10.Exit

Enter ur choice :6

Max element=30

BINARY SEARCH TREE

Main Menu

1.Create

2.Insert

3.Delete

4.Find

5.FindMin

6.FindMax

7.Inorder

8.Preorder

9.Postorder

10.Exit

Enter ur choice :7

10 20 25 30

BINARY SEARCH TREE

Main Menu

1.Create

2.Insert

3.Delete

4.Find

5.FindMin

6.FindMax

7.Inorder

8.Preorder

9.Postorder

10.Exit

Enter ur choice :8

10 20 30 25

BINARY SEARCH TREE

Main Menu

1.Create

2.Insert

3.Delete

4.Find

5.FindMin

6.FindMax

7.Inorder

8.Preorder

9.Postorder

10.Exit

Enter ur choice :9

25 30 20 10

BINARY SEARCH TREE

Main Menu

1.Create

2.Insert

3.Delete

4.Find

5.FindMin

6.FindMax

7.Inorder

8.Preorder

9.Postorder

10.Exit

Enter ur choice :3

Enter the data:10

20 25 30

BINARY SEARCH TREE

Main Menu

1.Create

2.Insert

3.Delete

4.Find

5.FindMin

6.FindMax

7.Inorder

8.Preorder

9.Postorder

10.Exit

Enter ur choice :10

RESULT:

The given program is implemented, executed, tested and verified successfully

EX.NO:7**IMPLEMENTATION OF AVL TREES****DATE:****AIM:**

To write a C program to implement AVL trees.

ALGORITHM:

Step 1: Create an AVL tree with defined number of nodes.

Step 1: Perform Insertion, Deletion operations

Step 1: Perform tree traversal

Step 1: Perform AVL tree rotations

PROGRAM:

```
#include<stdio.h>
typedef struct node
{
    int data;
    struct node *left,*right;
    int ht;
}node;

node *insert(node *,int);
node *Delete(node *,int);
void preorder(node *);
void inorder(node *);
int height( node *);
node *rotateright(node *);
node *rotateleft(node *);
node *RR(node *);
node *LL(node *);
node *LR(node *);
node *RL(node *);
int BF(node *);

int main()
{
    node *root=NULL;
    int x,n,i,op;

    do
    {
        printf("\n1)Create:");
        printf("\n2)Insert:");
        printf("\n3)Delete:");
        printf("\n4)Print:");
        printf("\n5)Quit:");
        printf("\n\nEnter Your Choice:");
        scanf("%d",&op);
```

```

switch(op)
{
    case 1: printf("\nEnter no. of elements:");
        scanf("%d",&n);
        printf("\nEnter tree data:");
        root=NULL;
        for(i=0;i<n;i++)
        {
            scanf("%d",&x);
            root=insert(root,x);
        }
        break;

    case 2: printf("\nEnter a data:");
        scanf("%d",&x);
        root=insert(root,x);
        break;

    case 3: printf("\nEnter a data:");
        scanf("%d",&x);
        root=Delete(root,x);
        break;

    case 4: printf("\nPreorder sequence:\n");
        preorder(root);
        printf("\n\nInorder sequence:\n");
        inorder(root);
        printf("\n\n");
        break;
}
}while(op!=5);

return 0;
}

node * insert(node *T,int x)
{
    if(T==NULL)
    {
        T=(node*)malloc(sizeof(node));
        T->data=x;
        T->left=NULL;
        T->right=NULL;
    }
    else
        if(x > T->data)    // insert in right subtree
        {
            T->right=insert(T->right,x);
        }

```

```

        if(BF(T)==-2)
            if(x>T->right->data)
                T=RR(T);
            else
                T=RL(T);
        }
    else
        if(x<T->data)
        {
            T->left=insert(T->left,x);
            if(BF(T)==2)
                if(x < T->left->data)
                    T=LL(T);
                else
                    T=LR(T);
        }

    T->ht=height(T);

    return(T);
}

node * Delete(node *T,int x)
{
    node *p;

    if(T==NULL)
    {
        return NULL;
    }
    else
        if(x > T->data)    // insert in right subtree
        {
            T->right=Delete(T->right,x);
            if(BF(T)==2)
                if(BF(T->left)>=0)
                    T=LL(T);
                else
                    T=LR(T);
        }
    else
        if(x<T->data)
        {
            T->left=Delete(T->left,x);
            if(BF(T)==-2) //Rebalance during windup
                if(BF(T->right)<=0)
                    T=RR(T);
                else

```



```

        T=RL(T);
    }
    else
    {
        //data to be deleted is found
        if(T->right!=NULL)
        { //delete its inorder succesor
            p=T->right;

            while(p->left!= NULL)
                p=p->left;

            T->data=p->data;
            T->right=Delete(T->right,p->data);

            if(BF(T)==2)//Rebalance during windup
                if(BF(T->left)>=0)
                    T=LL(T);
                else
                    T=LR(T);\
            }
            else
                return(T->left);
        }
    }
    T->ht=height(T);
    return(T);
}

```

```

int height(node *T)
{
    int lh,rh;
    if(T==NULL)
        return(0);

    if(T->left==NULL)
        lh=0;
    else
        lh=1+T->left->ht;

    if(T->right==NULL)
        rh=0;
    else
        rh=1+T->right->ht;

    if(lh>rh)
        return(lh);

    return(rh);
}

```

```

}

node * rotateright(node *x)
{
    node *y;
    y=x->left;
    x->left=y->right;
    y->right=x;
    x->ht=height(x);
    y->ht=height(y);
    return(y);
}

node * rotateleft(node *x)
{
    node *y;
    y=x->right;
    x->right=y->left;
    y->left=x;
    x->ht=height(x);
    y->ht=height(y);

    return(y);
}

node * RR(node *T)
{
    T=rotateleft(T);
    return(T);
}

node * LL(node *T)
{
    T=rotateright(T);
    return(T);
}

node * LR(node *T)
{
    T->left=rotateleft(T->left);
    T=rotateright(T);

    return(T);
}

node * RL(node *T)
{
    T->right=rotateright(T->right);

```

```

    T=rotateleft(T);
    return(T);
}

int BF(node *T)
{
    int lh,rh;
    if(T==NULL)
        return(0);

    if(T->left==NULL)
        lh=0;
    else
        lh=1+T->left->ht;

    if(T->right==NULL)
        rh=0;
    else
        rh=1+T->right->ht;

    return(lh-rh);
}

void preorder(node *T)
{
    if(T!=NULL)
    {
        printf("%d(Bf=%d)",T->data,BF(T));
        preorder(T->left);
        preorder(T->right);
    }
}

void inorder(node *T)
{
    if(T!=NULL)
    {
        inorder(T->left);
        printf("%d(Bf=%d)",T->data,BF(T));
        inorder(T->right);
    }
}

```

Output:

- 1)Create:
- 2)Insert:
- 3)Delete:

4) Print:
 5) Quit:
 Enter Your Choice:1
 Enter no. of elements:4
 Enter tree data:7 12 4 9
 1) Create:
 2) Insert:
 3) Delete:
 4) Print:
 5) Quit:
 Enter Your Choice:4
 Preorder sequence:
 7(Bf=-1)4(Bf=0)12(Bf=1)9(Bf=0)
 Inorder sequence:
 4(Bf=0)7(Bf=-1)9(Bf=0)12(Bf=1)
 1) Create:
 2) Insert:
 3) Delete:
 4) Print:
 5) Quit:
 Enter Your Choice:3
 Enter a data:7
 1) Create:
 2) Insert:
 3) Delete:
 4) Print:
 5) Quit:
 Enter Your Choice:4
 Preorder sequence:
 9(Bf=0)4(Bf=0)12(Bf=0)
 Inorder sequence:
 4(Bf=0)9(Bf=0)12(Bf=0)
 1) Create:
 2) Insert:
 3) Delete:
 4) Print:
 5) Quit:
 Enter Your Choice:5

RESULT:

Thus the C program to implement AVL trees has been executed and the output is verified.

EX.NO:8 IMPLEMENTATION OF PRIORITY QUEUE USING HEAPS**DATE:****AIM:**

To implement priority queue using heaps.

ALGORITHM:

Step 1: Start the Program

Step 2: heap is a binary tree with two important properties:

- For any node n other than the root, $n.key \geq n.parent.key$. In other words, the parent always has more priority than its children.
- If the heap has height h , the first $h-1$ levels are full, and on the last level the nodes are all packed to the left.

Step 3: implement the queue as a linked list, the element with most priority will be the first element of the list, so retrieving the content as well as removing this element are both $O(1)$ operations. However, inserting a new object in its right position requires traversing the list element by element, which is an $O(n)$ operation.

Step 4: Insert Element in Queue void insert (Object o, int priority) - inserts in the queue the specified object with the specified priority
 Algorithm insert (Object o, int priority)

Input: An object and the corresponding priority

Output: The object is inserted in the heap with the corresponding priority

lastNode \leftarrow getLast() //get the position at which to insert

lastNode.setKey(priority)

lastnode.setContent(o)

n lastNode

while n.getParent() != null and n.getParent().getKey() > priority

swap(n,n.getParent())

Step 5: Object DeleteMin() - removes from the queue the object with most priority

Algorithm removeMin()

lastNode <- getLast()

value lastNode.getContent()

swap(lastNode, root)

update lastNode

return value

PROGRAM:

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

#include<stdlib.h>

#include<process.h>

struct heapnode

```
{
    int capacity;
    int size;
    int *elements;
```

};

int isFull(struct heapnode *h)

```
{
    if(h->capacity==h->size)
        return 1;
    else
```

```

        return 0;
    }

int isEmpty(struct heapnode *h)
{
    if(h->size==0)
        return 1;
    else
        return 0;
}

void display(struct heapnode *h)
{
    printf("\nPriority Queue Display :");
    if(isEmpty(h))
    {
        printf("\nPriority queue is empty");
        return;
    }
    else

    for(int i=1;i<=h->size;i++)
        printf("%d\t",h->elements[i]);

}
struct heapnode * initialize()
{
    struct heapnode *t;
    int maxelements;
    printf("\nEnter the Size of the Priority queue :");
    scanf("%d",&maxelements);
    if(maxelements<5)
    {
        printf("Priority queue size is to small");
        getch();
        exit(0);
    }
    t=(struct heapnode *)malloc(sizeof(struct heapnode *));
    if(t==NULL)
    {
        printf("out of space!");
        getch();
        exit(0);
    }
    t->elements=(int *)malloc((maxelements+1)*sizeof(int));

    if(t->elements==NULL)
    {

```

```
printf("Out of space");
```

```

        getch();
        exit(0);
    }
    t->capacity=maxelements;
    t->size=0;
    t->elements=0;
    return t;
}
void insert(int x,struct heapnode *h)
{
    int i;
    if(isFull(h))
    {
        printf("Priority queue is full");
        return;
    }
    for(i=++h->size;h->elements[i/2]>x;i/=2)
        h->elements[i]=h->elements[i/2];
    h->elements[i]=x;
}
int deleteMin(struct heapnode *h)
{
    int i,child;
    int MinElement,LastElement;
    if(isEmpty(h))
    {
        printf("Priority queue is empty");
        return 0;
    }
    MinElement=h->elements[1];
    LastElement=h->elements[h->size--];
    for(i=1;i*2<=h->size;i=child)
    {
        child=i*2;
        if(child!=h->size&&h->elements[child+1]<h->elements[child])
            child++;
        if(LastElement>h->elements[child])
            h->elements[i]=h->elements[child];
        else
            break;
    }

    h->elements[i]=LastElement;
    return MinElement;
}
void main()
{
    int ch,ins,del;
    struct heapnode *h;

```



```

clrscr();
printf("\nPriority Queue using Heap");
h=initialize();
while(1)
{
    printf("\n1. Insert\n2. DeleteMin\n3. Display\n4. Exit");
    printf("\nEnter u r choice :");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:
            printf("\nEnter the element:");
            scanf("%d",&ins);
            insert(ins,h);
            break;
        case 2:
            del=deleteMin(h);
            printf("\nDeleted element is %d",del);
            getch();
            break;
        case 3:
            display(h);
            getch();
            break;
        case 4:
            exit(0);
    }
}
}

```

OUTPUT:

Priority Queue using Heap

Enter the Size of the Priority queue :14

1. Insert
2. DeleteMin
3. Display
4. Exit

Enter u r choice :1

Enter the element:10

1. Insert
2. DeleteMin
3. Display
4. Exit

Enter u r choice :1

Enter the element:34

1. Insert
2. DeleteMin
3. Display

4. Exit

```
Enter u r choice :1
Enter the element:24
1. Insert
2. DeleteMin
3. Display
4. Exit
Enter u r choice :1
Enter the element:67
1. Insert
2. DeleteMin
3. Display
4. Exit
Enter u r choice :3
Priority Queue Display :10    34    24    67
1. Insert
2. DeleteMin
3. Display
4. Exit
Enter u r choice :2
Deleted element is 10
1. Insert
2. DeleteMin
3. Display
4. Exit
Enter u r choice :2
Deleted element is 24
1. Insert
2. DeleteMin
3. Display
4. Exit
Enter u r choice :3
Priority Queue Display :34    67
1. Insert
2. DeleteMin
3. Display
4. Exit
Enter u r choice :4
```

Result:

Thus the 'C' program to implement priority queue using heaps has been executed and the output is verified

EX.NO:9A GRAPH REPRESENTATION AND TRAVERSAL ALGORITHM

DATE: DEPTH FIRST SEARCH

Aim:

To write a C program for representing the graph and traversing it using DFS.

Algorithm:

Step 1: Start from any vertex, say V_i .

Step 2: V_i is visited and then all vertices adjacent to V_i are traversed recursively using DFS.

Step 3: Since, a graph can have cycles. We must avoid revisiting a node. To do this, when we visit a vertex V, we mark it visited.

Step 4: A node that has already been marked as visited should not be selected for traversal.

Step 5: Marking of visited vertices can be done with the help of a global array `visited[]`.

Step 6: Array visited[] is initialized to false (0).

PROGRAM:

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

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

```
typedef struct node
```

```
{
    struct node *next;
    int vertex;
}node;
```

```
node *G[20];
```

```
//heads of linked list
```

```
int visited[20];
```

```
int n;
```

```
void read_graph();
```

```
//create adjacency list
```

```
void insert(int,int);
```

```
//insert an edge (vi,vj) in te adjacency list
```

```
void DFS(int);
```

```
void main()
```

```
{
    int i;
    read_graph();
    //initialised visited to 0
```

```
for(i=0;i<n;i++)
    visited[i]=0;
```

```

    DFS(0);
}

void DFS(int i)
{
    node *p;

    printf("\n%d",i);
    p=G[i];
    visited[i]=1;
    while(p!=NULL)
    {
        i=p->vertex;

        if(!visited[i])
            DFS(i);
        p=p->next;
    }
}

void read_graph()
{
    int i,vi,vj,no_of_edges;
    printf("Enter number of vertices:");

    scanf("%d",&n);

    //initialise G[] with a null

    for(i=0;i<n;i++)
    {
        G[i]=NULL;
        //read edges and insert them in G[]

        printf("Enter number of edges:");
        scanf("%d",&no_of_edges);

        for(i=0;i<no_of_edges;i++)
        {
            printf("Enter an edge(u,v):");
            scanf("%d%d",&vi,&vj);
            insert(vi,vj);
        }
    }
}

void insert(int vi,int vj)

```

```

{
    node *p,*q;

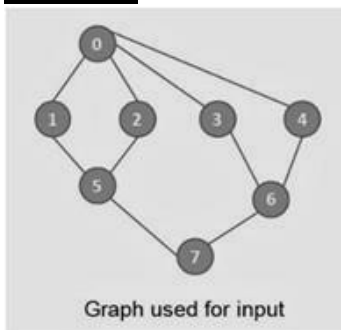
    //acquire memory for the new node
    q=(node*)malloc(sizeof(node));
    q->vertex=vj;
    q->next=NULL;

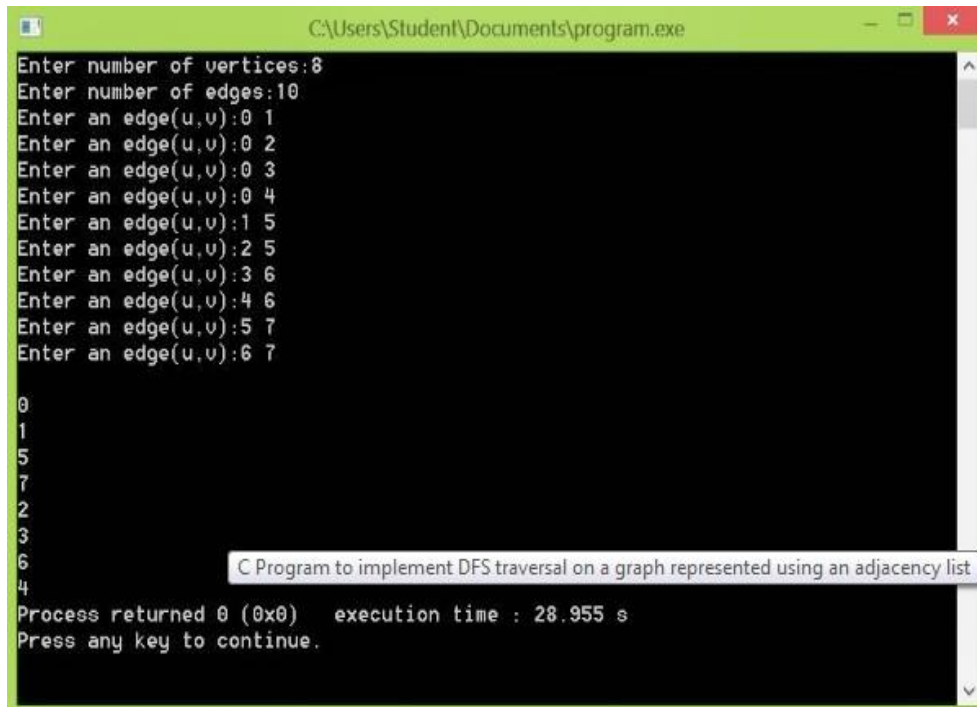
    //insert the node in the linked list number vi
    if(G[vi]==NULL)
        G[vi]=q;
    else
    {
        //go to end of the linked list
        p=G[vi];

        while(p->next!=NULL)
            p=p->next;
        p->next=q;
    }
}

```

OUTPUT:





The screenshot shows a Windows command prompt window titled "C:\Users\Student\Documents\program.exe". The program prompts the user to enter the number of vertices (8) and the number of edges (10). It then asks for 10 edges, each represented as (u, v). The edges entered are: (0, 1), (0, 2), (0, 3), (0, 4), (1, 5), (2, 5), (3, 6), (4, 6), (5, 7), and (6, 7). After the input, the program outputs the DFS traversal sequence: 0, 1, 5, 7, 2, 3, 6, 4. A tooltip is visible over the output, stating "C Program to implement DFS traversal on a graph represented using an adjacency list". The program concludes with "Process returned 0 (0x0) execution time : 28.955 s" and "Press any key to continue."

```
C:\Users\Student\Documents\program.exe
Enter number of vertices:8
Enter number of edges:10
Enter an edge(u,v):0 1
Enter an edge(u,v):0 2
Enter an edge(u,v):0 3
Enter an edge(u,v):0 4
Enter an edge(u,v):1 5
Enter an edge(u,v):2 5
Enter an edge(u,v):3 6
Enter an edge(u,v):4 6
Enter an edge(u,v):5 7
Enter an edge(u,v):6 7

0
1
5
7
2
3
6
4
C Program to implement DFS traversal on a graph represented using an adjacency list
Process returned 0 (0x0) execution time : 28.955 s
Press any key to continue.
```

RESULT:

Thus the C program for representing the graph and traversing it using DFS has been executed and the output is verified


```

void BFS(int v)
{
    int i;
    insert_queue(v);
    state[v] = waiting;
    while(!isEmpty_queue())
    {
        v = delete_queue( );
        printf("%d ",v);
        state[v] = visited;
        for(i=0; i<n; i++)
        {
            if(adj[v][i] == 1 && state[i] == initial)
            {
                insert_queue(i);
                state[i] = waiting;
            }
        }
        printf("\n");
    }
}

void insert_queue(int vertex)
{
    if(rear == MAX-1)
        printf("Queue Overflow\n");
    else
    {
        if(front == -1)
            front = 0;
        rear = rear+1;
        queue[rear] = vertex ;
    }
}

int isEmpty_queue()
{
    if(front == -1 || front > rear)
        return 1;
    else
        return 0;
}

int delete_queue()
{
    int delete_item;
    if(front == -1 || front > rear)
    {
        printf("Queue Underflow\n");
        exit(1);
    }
}

```

```
    delete_item = queue[front];
    front = front+1;
    return delete_item;
}
void create_graph()
{
    int count,max_edge,origin,destin;
    printf("Enter number of vertices : ");
    scanf("%d",&n);
    max_edge = n*(n-1);
    for(count=1; count<=max_edge; count++)
    {
        printf("Enter edge %d( -1 -1 to quit ) : ",count);
        scanf("%d %d",&origin,&destin);
        if((origin == -1) && (destin == -1))
            break;
        if(origin>=n || destin>=n || origin<0 || destin<0)
        {
            printf("Invalid edge!\n");
            count--;
        }
        else
        {
            adj[origin][destin] = 1;
        }
    }
}
```

OUTPUT:

```

tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$ ./a.out
Enter number of vertices : 9
Enter edge 1( -1 -1 to quit ) : 0
1
Enter edge 2( -1 -1 to quit ) : 0
3
Enter edge 3( -1 -1 to quit ) : 0
4
Enter edge 4( -1 -1 to quit ) : 1
2
Enter edge 5( -1 -1 to quit ) : 3
6
Enter edge 6( -1 -1 to quit ) : 4
7
Enter edge 7( -1 -1 to quit ) : 6
4
Enter edge 8( -1 -1 to quit ) : 6
7
Enter edge 9( -1 -1 to quit ) : 2
5
Enter edge 10( -1 -1 to quit ) : 4
5
Enter edge 11( -1 -1 to quit ) : 7
5
Enter edge 12( -1 -1 to quit ) : 7
8
Enter edge 13( -1 -1 to quit ) : -1
-1
Enter Start Vertex for BFS:
0
0 1 3 4 2 6 5 7 8
tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$

```

RESULT:

Thus the C program for representing the graph and traversing it using BFS has been executed and the output is verified.

Ex.No:10**APPLICATIONS OF GRAPHS – DIJKSTRA'S ALGORITHM****DATE:****AIM:**

To implement Dijkstra's algorithm using priority queues.

ALGORITHM:

1. Assign to every node a distance value. Set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes as unvisited. Set initial node as current.
3. For current node, consider all its unvisited neighbors and calculate their distance (from the initial node). For example, if current node (A) has distance of 6, and an edge connecting it with another node (B) is 2, the distance to B through A will be $6+2=8$. If this distance is less than the previously recorded distance (infinity in the beginning, zero for the initial node), overwrite the distance.
4. When we are done considering all neighbors of the current node, mark it as visited. A visited node will not be checked ever again; its distance recorded now is final and minimal.
5. Set the unvisited node with the smallest distance (from the initial node) as the next "current node" and continue from step 3 .

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int graph[15][15],s[15],pathestimate[15],mark[15];
    int num_of_vertices,source,i,j,u,predecessor[15];
    int count=0;
    int minimum(int a[],int m[],int k);
    void printpath(int,int,int[]);
    printf("\nenter the no.of vertices\n");
    scanf("%d",&num_of_vertices);
    if(num_of_vertices<=0)
    {
        printf("\nthis is meaningless\n");
        exit(1);
    }
    printf("\nenter the adjacent matrix\n");
    for(i=1;i<=num_of_vertices;i++)
    {
        printf("\nenter the elements of row %d\n",i);
        for(j=1;j<=num_of_vertices;j++)
        {
            scanf("%d",&graph[i][j]);
        }
    }
}
```

```

printf("\nEnter the source vertex\n");
scanf("%d",&source);
for(j=1;j<=num_of_vertices;j++)
{
    mark[j]=0;
    pathestimate[j]=999;
    predecessor[j]=0;
}
pathestimate[source]=0;

while(count<num_of_vertices)
{
    u=minimum(pathestimate,mark,num_of_vertices);
    s[++count]=u;
    mark[u]=1;

    for(i=1;i<=num_of_vertices;i++)
    {
        if(graph[u][i]>0)
        {
            if(mark[i]!=1)
            {
                if(pathestimate[i]>pathestimate[u]+graph[u][i])
                {
                    pathestimate[i]=pathestimate[u]+graph[u][i];
                    predecessor[i]=u;
                }
            }
        }
    }
}

for(i=1;i<=num_of_vertices;i++)
{
    printpath(source,i,predecessor);
    if(pathestimate[i]!=999)
        printf("->(%d)\n",pathestimate[i]);
}

int minimum(int a[],int m[],int k)
{
    int mi=999;
    int i,t;
    for(i=1;i<=k;i++)
    {

```

```

        if(m[i]!=1)
        {
            if(mi>=a[i])
            {
                mi=a[i];
                t=i;
            }
        }
    }
    return t;
}

void printpath(int x,int i,int p[])
{
    printf("\n");
    if(i==x)
    {
        printf("%d",x);
    }
    else if(p[i]==0)
        printf("no path from %d to %d",x,i);
    else
    {
        printpath(x,p[i],p);
        printf("..%d",i);
    }
}

```

OUTPUT:

```

enter the no.of vertices
2
enter the adjacent matrix
enter the elements of row 1
1
2
enter the elements of row 2
2
3
enter the source vertex
1
1->(0)
1..2->(2)

```

RESULT:

Thus the C program for implementing Dijkstra's algorithm has been executed and the output is verified.

EX.NO:11.A IMPLEMENTATION OF SEARCHING AND SORTING ALGORITHMS**DATE:****BINARY SEARCH ALGORITHM****AIM:** To write a C program to implement binary search.**ALGORITHM:**

Step 1: Input a sorted LIST of size N and Target Value T

OUTPUT: Position of T in the LIST = I

Step 2: If MAX = N and MIN = 1 set FOUND = false

Step 3: WHILE (FOUND is false) and (MAX >= MIN) Set

MID = (MAX + MIN) DIV 2

If T = LIST [MID]

I=MID

FOUND = true

Step 4: Else If T < LIST[MID] Set MAX = MID-1

Step 5: Else MIN = MD+1

Step 6: END

PROGRAM:

#include <stdio.h>

int main()

{

int c, first, last, middle, n, search, array[100];

printf("Enter number of elements\n");

scanf("%d",&n);

printf("Enter %d integers\n", n);

for (c = 0 ; c < n ; c++)

scanf("%d",&array[c]);

printf("Enter value to find\n");

scanf("%d",&search);

first = 0;

last = n - 1;

middle = (first+last)/2;

while(first <= last)

{

if (array[middle] < search)

first = middle + 1;

else if (array[middle] == search)

{

printf("%d found at location %d.\n", search, middle+1);

break;

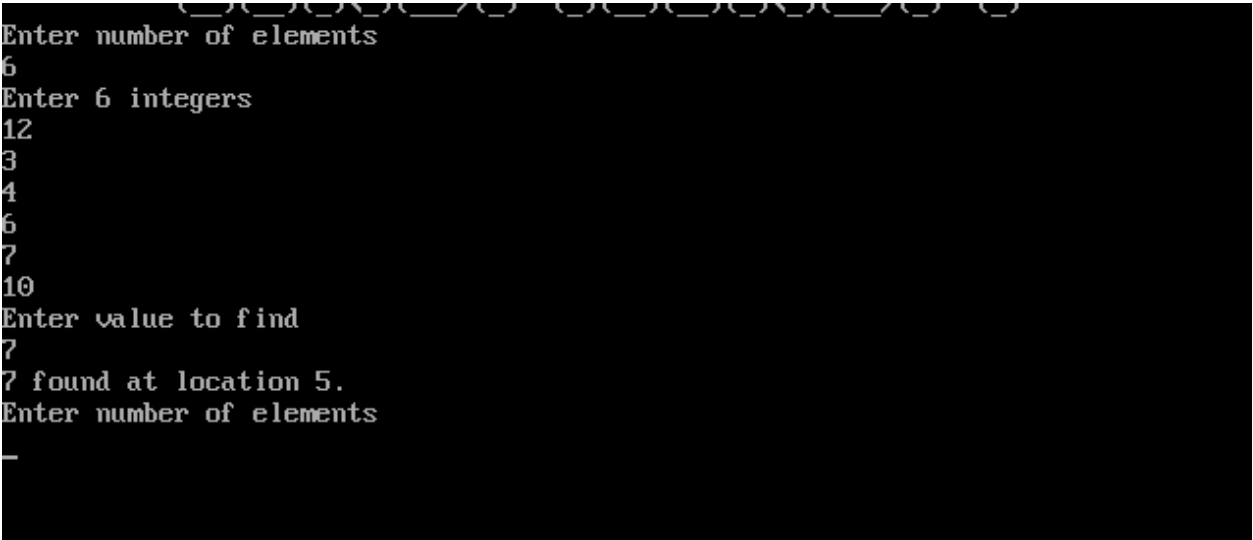
}

} else

```
        last = middle - 1;

        middle = (first + last)/2;
    }
    if ( first > last )
        printf("Not found! %d is not present in the list.\n", search);

    return 0;
}
```

OUTPUT:

```
Enter number of elements
6
Enter 6 integers
12
3
4
6
7
10
Enter value to find
7
7 found at location 5.
Enter number of elements
-
```

RESULT:

Thus the C program for implementing Binary Search algorithm has been executed and the output is verified.

EX.NO:11.B IMPLEMENTATION OF SEARCHING AND SORTING ALGORITHMS**DATE: LINEAR SEARCH ALGORITHM****AIM:**

To write a C program to implement LINEAR search.

ALGORITHM

Linear search is implemented using following

- 1: Read the search element from the use
- 2: Compare, the search element with the first element in the list.
- 3: If both are matching, then display "Given element found!!!" and terminate the function
- 4: If both are not matching, then compare search element with the next element in the list.
- 5: Repeat s 3 and 4 until the search element is compared with the last element in the list.
- 6: If the last element in the list is also doesn't match, then display "Element not found!!!" and terminate the function.

PROGRAM

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

void main(){
    int list[20],size,i,sElement;

    printf("Enter size of the list: ");
    scanf("%d",&size);

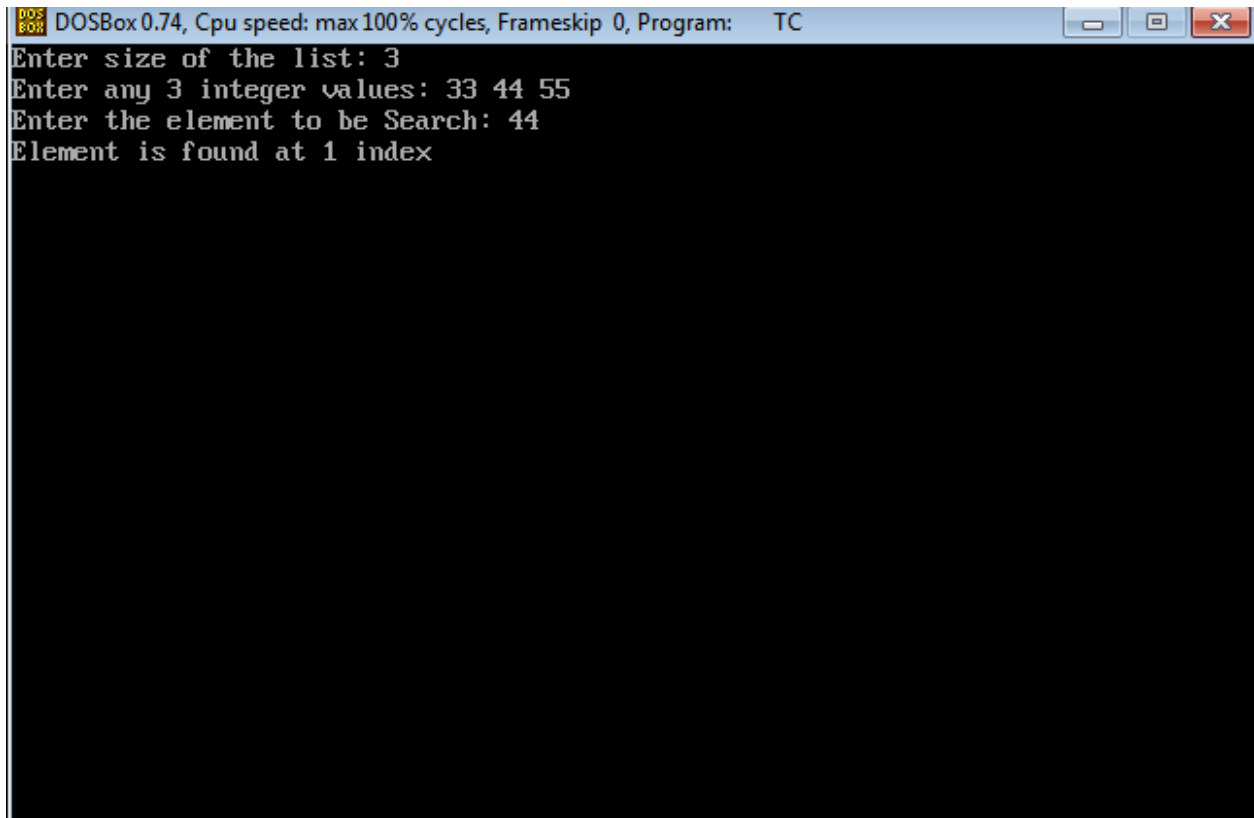
    printf("Enter any %d integer values: ",size);
    for(i = 0; i < size; i++)
        scanf("%d",&list[i]);

    printf("Enter the element to be Search: ");
    scanf("%d",&sElement);

    // Linear Search Logic
    for(i = 0; i < size; i++)
    {
        if(sElement == list[i])
        {
            printf("Element is found at %d index", i);
            break;
        }
    }
    if(i == size)
        printf("Given element is not found in the list!!!");
    getch();
}
```

```
}
```

OUTPUT



The screenshot shows a DOSBox window titled "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The window contains a black terminal area with white text. The text shows the program's prompts and user input: "Enter size of the list: 3", "Enter any 3 integer values: 33 44 55", "Enter the element to be Search: 44", and the output "Element is found at 1 index".

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter size of the list: 3
Enter any 3 integer values: 33 44 55
Enter the element to be Search: 44
Element is found at 1 index
```

RESULT:

Thus the C program for implementing Binary Search algorithm has been executed and the output is verified.

EX.NO:11.C IMPLEMENTATION OF SEARCHING AND SORTING ALGORITHMS

DATE: INSERTION SORT

AIM:

To write a C program to implement Insertion Sort.

ALGORITHM:

Step 1: The second element of an array is compared with the elements that appears before it (only first element in this case). If the second element is smaller than first element, second element is inserted in the position of first element. After first step, first two elements of an array will be sorted.

Step 2: The third element of an array is compared with the elements that appears before it (first and second element). If third element is smaller than first element, it is inserted in the position of first element. If third element is larger than first element but, smaller than second element, it is inserted in the position of second element. If third element is larger than both the elements, it is kept in the position as it is. After second step, first three elements of an array will be sorted.

Step 3: Similarly, the fourth element of an array is compared with the elements that appears before it (first, second and third element) and the same procedure is applied and that element is inserted in the proper position. After third step, first four elements of an array will be sorted.

PROGRAM:

```
#include<stdio.h>
int main()
{
    int data[100],n,temp,i,j;
    printf("Enter number of terms(should be less than 100): ");
    scanf("%d",&n);
    printf("Enter elements: ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&data[i]);
    }
    for(i=1;i<n;i++)
    {
        temp = data[i];
        j=i-1;
        while(temp<data[j] && j>=0)
        /*To sort elements in descending order, change temp<data[j] to temp>data[j] in above line.*/
        {
            data[j+1] = data[j];
            --j;
        }
        data[j+1]=temp;
    }
}
```

```
    }  
    printf("In ascending order: ");  
    for(i=0; i<n; i++)  
        printf("%d\t",data[i]);  
    return 0;  
}
```

OUTPUT:

Enter number of terms(should be less than 100): 5

Enter elements: 12

1

2

5

3

In ascending order: 1 2 3 5 12

RESULT:

Thus the C program for implementing insertion sort has been executed and the output is verified.

EX.NO:11.D IMPLEMENTATION OF SEARCHING AND SORTING ALGORITHMS

DATE: SELECTION SORT

AIM:

To write a C program to implement Insertion Sort.

ALGORITHM:

The selection sort algorithm is performed using following

- 1: Select the first element of the list (i.e., Element at first position in the list).
- 2: Compare the selected element with all other elements in the list.
- 3: For every comparison, if any element is smaller than selected element (for Ascending order), then these two are swapped.
- 4: Repeat the same procedure with next position in the list till the entire list is sorted.

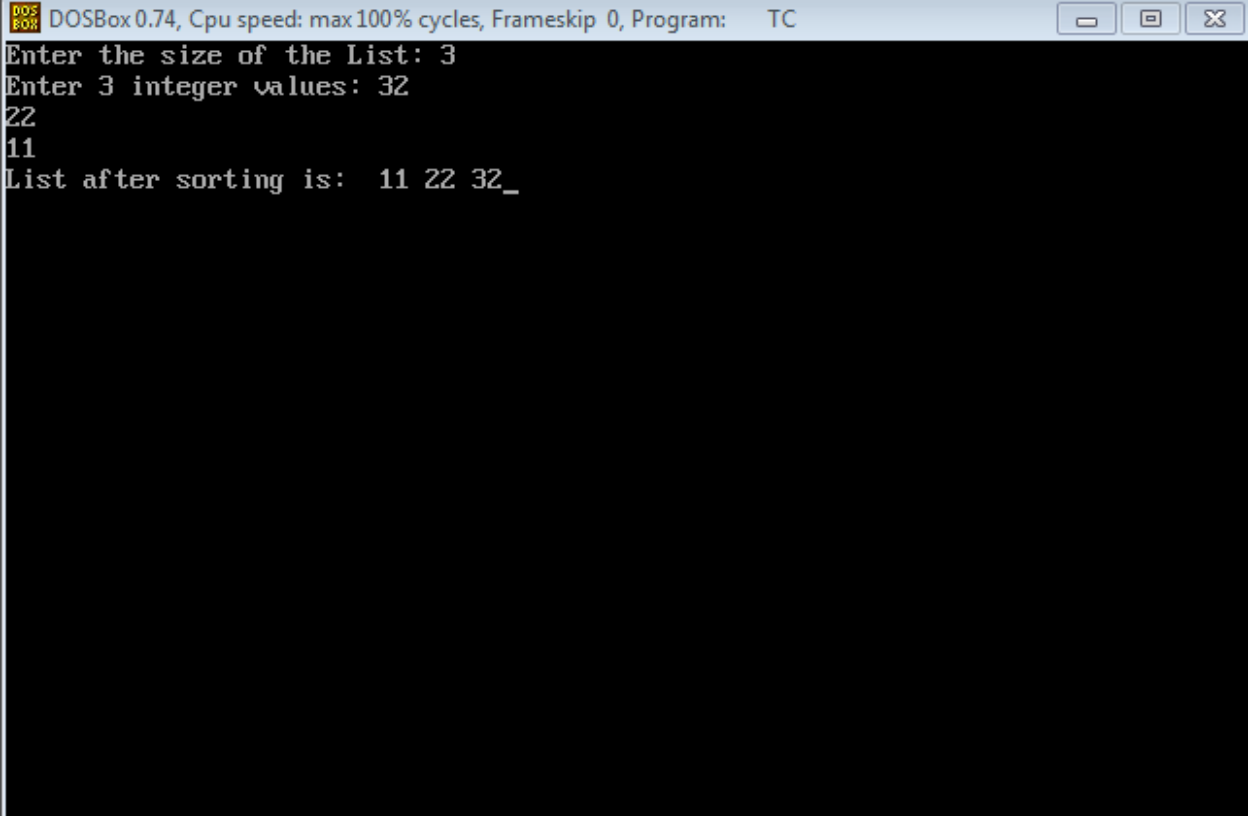
PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main(){
    int size,i,j,temp,list[100];
    clrscr();
    printf("Enter the size of the List: ");
    scanf("%d",&size);

    printf("Enter %d integer values: ",size);
    for(i=0; i<size; i++)
        scanf("%d",&list[i]);
    for(i=0; i<size; i++){
        for(j=i+1; j<size; j++){
            if(list[i] > list[j])
            {
                temp=list[i];
                list[i]=list[j];
                list[j]=temp;
            }
        }
    }

    printf("List after sorting is: ");
    for(i=0; i<size; i++)
        printf(" %d",list[i]);

    getch();
}
```

OUTPUT:A screenshot of a DOSBox window titled "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The window has a black background with white text. The text shows the program's input and output: "Enter the size of the List: 3", "Enter 3 integer values: 32", "22", "11", and "List after sorting is: 11 22 32_".

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter the size of the List: 3
Enter 3 integer values: 32
22
11
List after sorting is: 11 22 32_
```

RESULT:

Thus the C program for implementing insertion sort has been executed and the output is verified.

EX.NO:11.E IMPLEMENTATION OF SEARCHING AND SORTING ALGORITHMS**DATE:****BUBBLE SORT****AIM:**

To write a C program to implement Bubble Sort.

ALGORITHM:

- 1: Repeat s 2 and 3 for i=1 to 10
- 2: Set j=1
- 3: Repeat while j<=n
 - (A) if a[i] < a[j]
 - Then interchange a[i] and a[j]
 - [End of if]
 - (B) Set j = j+1
 [End of Inner Loop]
- [End of 1 Outer Loop]
- 4: Exit

PROGRAM:

```
#include <stdio.h>
#include<conio.h>
int main()
{
    int array[100], n, c, d, swap;

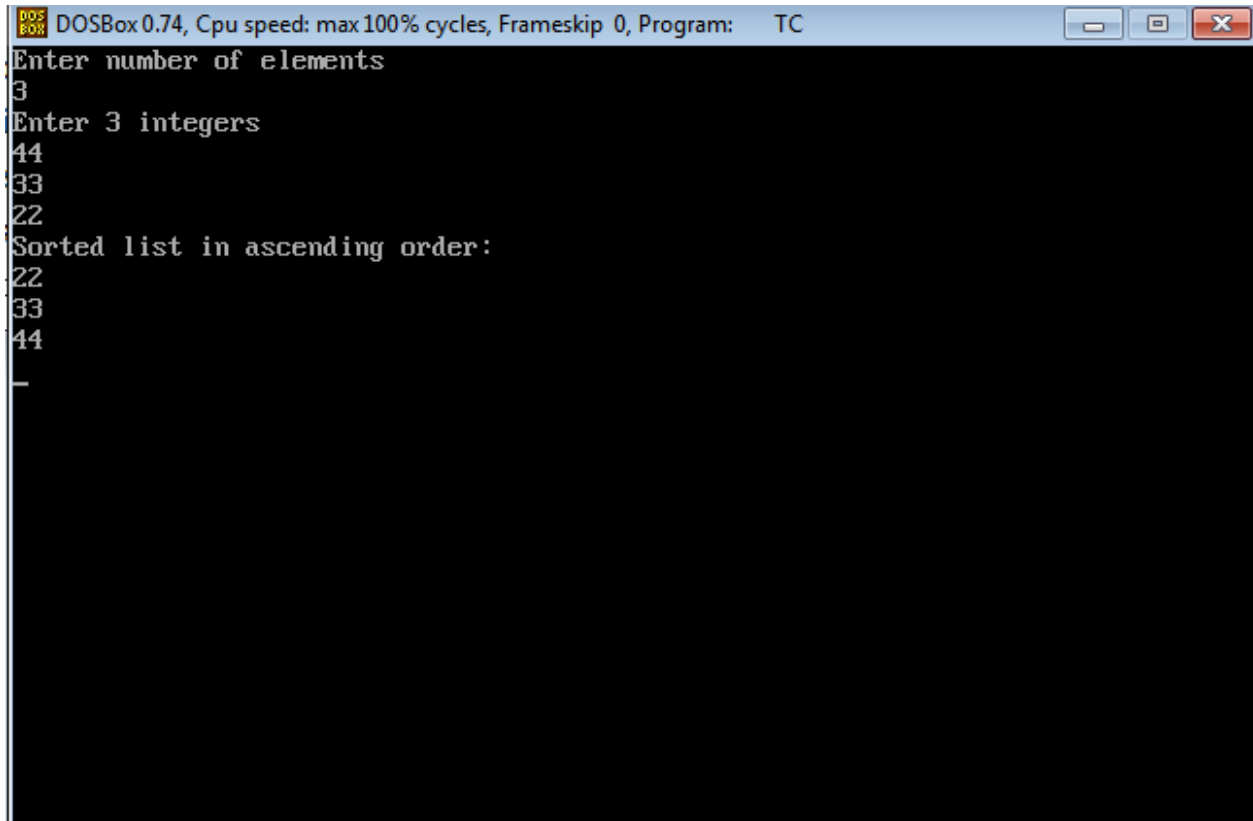
    printf("Enter number of elements\n");
    scanf("%d", &n);

    printf("Enter %d integers\n", n);

    for (c = 0; c < n; c++)
        scanf("%d", &array[c]);

    for (c = 0 ; c < n - 1; c++)
    {
        for (d = 0 ; d < n - c - 1; d++)
        {
            if (array[d] > array[d+1]) /* For decreasing order use < */
            {
                swap    = array[d];
                array[d] = array[d+1];
                array[d+1] = swap;
            }
        }
    }
    printf("Sorted list in ascending order:\n");
    for (c = 0; c < n; c++)
```

```
printf("%d\n", array[c]);  
  
getch();  
}
```

OUTPUT:

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC  
Enter number of elements  
3  
Enter 3 integers  
44  
33  
22  
Sorted list in ascending order:  
22  
33  
44  
-
```

RESULT:

Thus the C program for implementing insertion sort has been executed and the output is verified.

EX.NO:11.F IMPLEMENTATION OF SEARCHING AND SORTING ALGORITHMS**DATE: RADIX SORT****AIM:**

To write a C program to implement Radix Sort.

ALGORITHM:

1. Take the least significant digit of each key.
2. Group the keys based on that digit, but otherwise keep the original order of keys.
3. Repeat the grouping process with each more significant digit.
4. The sort in step 2 is usually done using bucket sort or counting sort, which are efficient in this case since there are usually only a small number of digits.

PROGRAM:

```
#include<stdio.h>
// Function to find largest element
int largest(int a[], int n)
{
    int large = a[0], i;
    for(i = 1; i < n; i++)
    {
        if(large < a[i])
            large = a[i];
    }
    return large;
}
// Function to perform sorting
void RadixSort(int a[], int n)
{
    int bucket[10][10], bucket_count[10];
    int i, j, k, remainder, NOP=0, divisor=1, large, pass;

    large = largest(a, n);
    printf("The large element %d\n", large);
    while(large > 0)
    {
        NOP++;
        large/=10;
    }

    for(pass = 0; pass < NOP; pass++)
    {
        for(i = 0; i < 10; i++)
        {
            bucket_count[i] = 0;
        }
        for(i = 0; i < n; i++)
```

```

    {
        remainder = (a[i] / divisor) % 10;
        bucket[remainder][bucket_count[remainder]] = a[i];
        bucket_count[remainder] += 1;
    }
    i = 0;
    for(k = 0; k < 10; k++)
    {
        for(j = 0; j < bucket_count[k]; j++)
        {
            a[i] = bucket[k][j];
            i++;
        }
    }
    divisor *= 10;
    for(i = 0; i < n; i++)
        printf("%d ",a[i]);
    printf("\n");
}
}
//program starts here
int main()
{
    int i, n, a[10];
    printf("Enter the number of elements :: ");
    scanf("%d",&n);
    printf("Enter the elements :: ");
    for(i = 0; i < n; i++)
    {
        scanf("%d",&a[i]);
    }
    RadixSort(a,n);
    printf("The sorted elements are :: ");
    for(i = 0; i < n; i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}

```

OUTPUT:

```

Enter the number of elements :: 7
Enter the elements :: 21 32 11 58 98 45 21
The large element 98
21 11 21 32 45 58 98
11 21 21 32 45 58 98
The sorted elements are :: 11 21 21 32 45 58 98

```

RESULT:

Thus the C program for implementing insertion sort has been executed and the output is verified.

EX.NO:11.G IMPLEMENTATION OF SEARCHING AND SORTING ALGORITHMS**DATE:****SHELL SORT****AIM:**

To write a C program to implement Shell Sort.

ALGORITHM:

1. Start sorting with the largest increment h_k
2. Sort all sub arrays of elements that are h_k apart so that $A[i] \leq A[i+h_k]$ for all i known as an h_k -sort
3. Go to next smaller increment h_{k-1} and repeat
4. Stop sorting after $h_1=1$.

PROGRAM

```
#include<stdio.h>
void ShellSort(int a[], int n)
{
    int i, j, increment, tmp;
    for(increment = n/2; increment > 0; increment /= 2)
    {
        for(i = increment; i < n; i++)
        {
            tmp = a[i];
            for(j = i; j >= increment; j -= increment)
            {
                if(tmp < a[j-increment])
                    a[j] = a[j-increment];
                else
                    break;
            }
            a[j] = tmp;
        }
    }
}
int main()
{
    int i, n, a[10];
    printf("Enter the number of elements :: ");
    scanf("%d",&n);
    printf("Enter the elements :: ");
    for(i = 0; i < n; i++)
    {
        scanf("%d",&a[i]);
    }
    ShellSort(a,n);
    printf("The sorted elements are :: ");
    for(i = 0; i < n; i++)
        printf("%d ",a[i]);
}
```

```
printf("\n");  
return 0;  
}
```

OUTPUT:

Enter the number of elements :: 6

Enter the elements :: 50 30 10 40 20 60

The sorted elements are :: 10 20 30 40 50 60

RESULT:

Thus the C program for implementing insertion sort has been executed and the output is verified

EX.NO:12**HASHING – ANY TWO COLLISION TECHNIQUES****DATE:****AIM:** To write a C program to implement Hashing with collision techniques.**ALGORITHM:**

Step 1 :Create an array of structure (i.e a hash table).

Step 2. Take a key and a value to be stored in hash table as input.

Step 3. Corresponding to the key, an index will be generated i.e every key is stored in a particular array index.

Step 4. Using the generated index, access the data located in that array index.

Step 5. In case of absence of data, create one and insert the data item (key and value) into it and increment the size of hash table.

Step 6. In case the data exists, probe through the subsequent elements (looping back if necessary) for free space to insert new data item.

Note: Here, unlike quadratic and linear probing, we will first calculate another hash code of same key using formula-: $\text{hashcode2} = \text{primeNum} - (\text{key} \% \text{primeNum})$, where primeNum is largest prime number less than array size

Probing formula after calculating hashcode2 -:

 $(\text{hashcode1} + (h * \text{hashcode2})) \% \text{arraySize}$, $h = 1, 2, 3, 4$ and so on

Step 7. To display all the elements of hash table, element at each index is accessed (via for loop).

Step 8. To remove a key from hash table, we will first calculate its index and delete it if key matches, else probe through elements (using above formula) until we find key or an empty space where not a single data has been entered (means data does not exist in the hash table).

PROGRAM:

#include<stdio.h>

#include<conio.h>

#include<math.h>

struct data

{

int key;

int value;

};

struct hashtable_item

{

int flag;

/*

* flag = 0 : data not present

* flag = 1 : some data already present

* flag = 2 : data was present, but deleted

*/

struct data *item;

};

```

struct hashtable_item *array;
int max = 7;
int size = 0;
int prime = 3;

int hashcode1(int key)
{
    return (key % max);
}

int hashcode2(int key)
{
    return (prime - (key % prime));
}

void insert(int key, int value)
{
    int hash1 = hashcode1(key);
    int hash2 = hashcode2(key);

    int index = hash1;

    /* create new data to insert */
    struct data *new_item = (struct data*) malloc(sizeof(struct data));
    new_item->key = key;
    new_item->value = value;

    if (size == max)
    {
        printf("\n Hash Table is full, cannot insert more items \n");
        return;
    }

    /* probing through other array elements */
    while (array[index].flag == 1) {
        if (array[index].item->key == key)
        {
            printf("\n Key already present, hence updating its value \n");
            array[index].item->value = value;
            return;
        }
        index = (index + hash2) % max;
    }
    if (index == hash1)
    {
        printf("\n Add is failed \n");
        return;
    }
}

```

```

        printf("\n probing \n");

    }

    array[index].item = new_item;
    array[index].flag = 1;
    size++;
    printf("\n Key (%d) has been inserted \n", key);

}

/* to remove an element from the array */
void remove_element(int key)
{
    int hash1 = hashcode1(key);
    int hash2 = hashcode2(key);
    int index = hash1;

    if (size == 0)
    {
        printf("\n Hash Table is empty \n");
        return;
    }

    /* probing through other elements */
    while (array[index].flag != 0)
    {
        if (array[index].flag == 1 && array[index].item->key == key)
        {
            array[index].item = NULL;
            array[index].flag = 2;
            size--;
            printf("\n Key (%d) has been removed \n", key);
            return;
        }
        index = (index + hash2) % max;
        if (index == hash1)
        {
            break;
        }
    }

    printf("\n Key (%d) does not exist \n", key);

}

int size_of_hashtable()
{

```

```

        return size;
    }
    /* displays all elements of array */
    void display()
    {
        int i;
        for (i = 0; i < max; i++)
        {
            if (array[i].flag != 1)
            {
                printf("\n Array[%d] has no elements \n", i);
            }
            else
            {
                printf("\n Array[%d] has elements \n Key (%d) and Value (%d) \n", i, array[i].item-
>key, array[i].item->value);
            }
        }
    }
    /* initializes array */
    void init_array()
    {
        int i;
        for(i = 0; i < max; i++)
        {
            array[i].item = NULL;
            array[i].flag = 0;
        }
        prime = get_prime();
    }
    /* returns largest prime number less than size of array */
    int get_prime()
    {
        int i,j;
        for (i = max - 1; i >= 1; i--)
        {
            int flag = 0;
            for (j = 2; j <= (int)sqrt(i); j++)
            {
                if (i % j == 0)
                {
                    flag++;
                }
            }
            if (flag == 0)
            {
                return i;
            }
        }
    }

```



```

    }
    return 3;
}
void main()
{
    int choice, key, value, n, c;
    clrscr();
    array = (struct hashtable_item*) malloc(max * sizeof(struct hashtable_item));
    init_array();
    do {
        printf("Implementation of Hash Table in C with Double Hashing.\n\n");
        printf("MENU:- \n1.Inserting item in the Hash Table"
            "\n2.Removing item from the Hash Table"
            "\n3.Check the size of Hash Table"
            "\n4.Display Hash Table"
            "\n\n Please enter your choice:-");

        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                printf("Inserting element in Hash Table\n");
                printf("Enter key and value-:\t");
                scanf("%d %d", &key, &value);
                insert(key, value);
                break;
            case 2:
                printf("Deleting in Hash Table \n Enter the key to delete:-");
                scanf("%d", &key);
                remove_element(key);
                break;
            case 3:
                n = size_of_hashtable();
                printf("Size of Hash Table is-:%d\n", n);

                break;
            case 4:
                display();
                break;
            default:

                printf("Wrong Input\n");
        }
        printf("\n Do you want to continue-:(press 1 for yes)\t");
        scanf("%d", &c);
    }while(c == 1);
    getch();}

```

OUTPUT:

MENU-:

1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice-: 3

Size of hash table is-: 0

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Double Hashing

MENU-:

1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice-: 1

Inserting element in hash table

Enter key and value-: 12 1 Key

(12) has been inserted

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Double Hashing

MENU-:

1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice-: 1

Inserting element in Hash Table

Enter key and value-: 47 1

probing

Key (47) has been inserted

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Double Hashing

MENU-:

1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice-: 1

Inserting element in Hash Table

Enter key and value-: 61 1

probing

Key (61) has been inserted

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Double Hashing

MENU-:

1. Inserting item in the Hash Table
2. Removing item from the Hash Table

3. Check the size of Hash Table

4. Display Hash Table

Please enter your choice-: 3

Size of hash table is-: 3

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Double Hashing

MENU-:

1. Inserting item in the Hash Table

2. Removing item from the Hash Table

3. Check the size of Hash Table

4. Display Hash Table

Please enter your choice-: 4

Array[0] has no elements

Array[1] has elements-:

47 (key) and 1 (value)

Array[2] has elements-:

61 (key) and 1 (value)

Array[3] has elements-:

Array[4] has no elements

Array[5] has elements-:

12 (key) and 1 (value)

Array[6] has no elements

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Double Hashing

MENU-:

1. Inserting item in the Hash Table

2. Removing item from the Hash Table

3. Check the size of Hash Table

4. Display Hash Table

Please enter your choice-: 2

Deleting in hash table

Enter the key to delete-: 61

Key (61) has been removed

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Double Hashing

MENU-:

1. Inserting item in the Hash Table

2. Removing item from the Hash Table

3. Check the size of Hash Table

4. Display a Hash Table

Please enter your choice-: 2

Deleting in hash table

Enter the key to delete-: 61

This key does not exist

Do you want to continue-:(press 1 for yes) 2

RESULT:

Thus the C program for implementing Hashing Techniques has been executed and the output is verified.